Pointer-Based String Processing

- Character constant
  - Enclosed in single quotes,
  - forexample: `'z'`
- A string is a series of characters treated as a single unit.
  - May include letters, digits and various special characters such as **+**, **-**, *****, **/** and **$**.
- A string is an array of characters ending with a null character (`'\0'`)
- String
  - Enclosed in double quotes,
  - forexample: `"I like C++"`

Pointer-Based String Processing

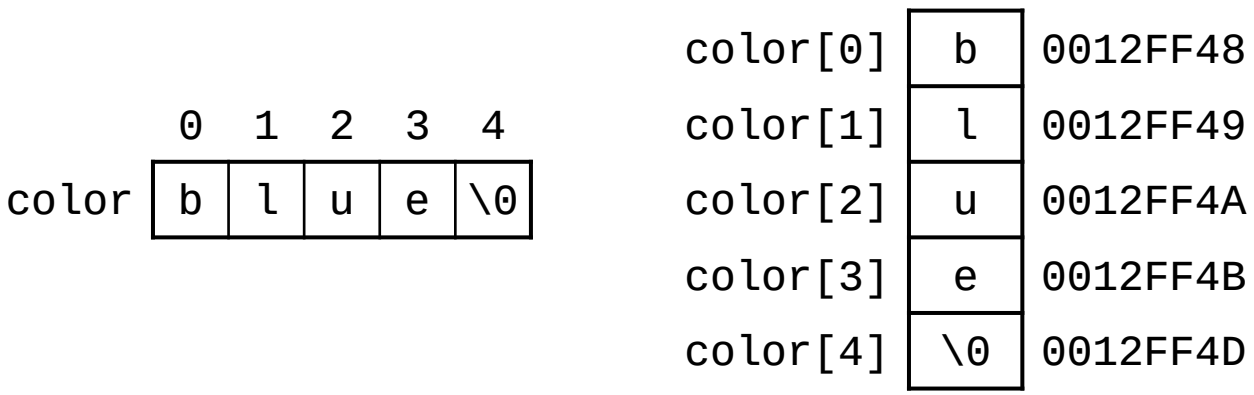
- All strings end with null (`'\0'`)
- Character array

```
char color[ 5 ] = "blue";
```

  - Creates 5 element **`char`** array **`color`**
  - Null character (`'\0'`) implicitly added

- Alternative forcharacter array

```
char color[ 5 ] = { 'b', 'l', 'u', 'e', '\0' };
```



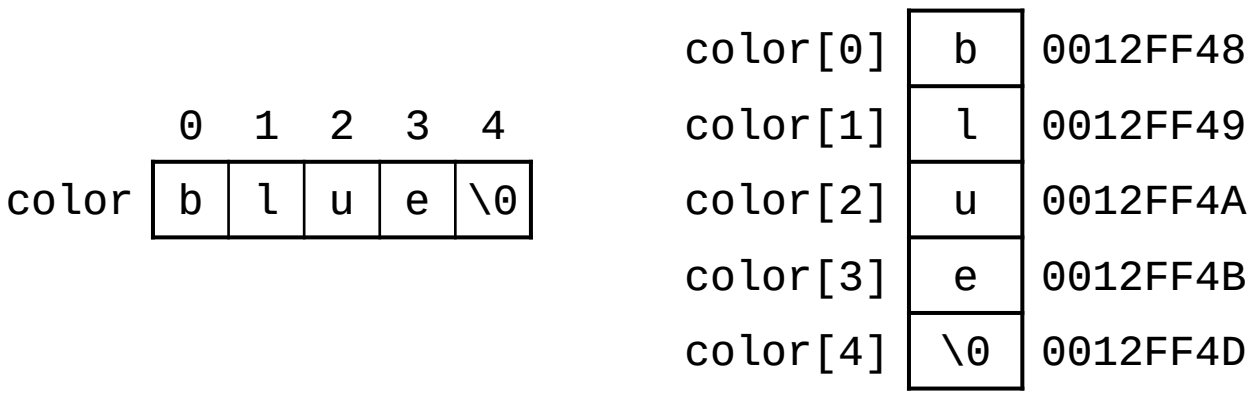
Pointer-Based String Processing

- All strings end with null (`'\0'`)
- Character array

```
char color[] = "blue";
```

  - Creates 5 element **`char`** array **`color`**
  - Null character (`'\0'`) implicitly added
- Alternative forcharacter array

```
char color[] = { 'b', 'l', 'u', 'e', '\0' };
```



Pointer-Based String Processing

- Input from keyboard

```
char color[ 10 ];
cin >> color;
```

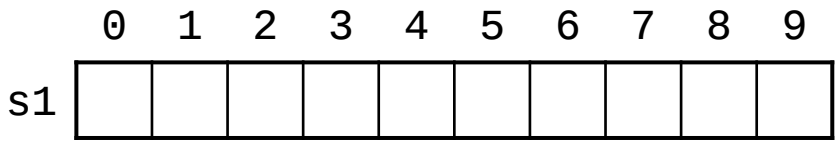
  - Puts user input in string
    - Stops at first whitespace character
    - Adds `null` character
  - If too much text entered, run time error
- Printing strings

```
cout << color << endl;
```

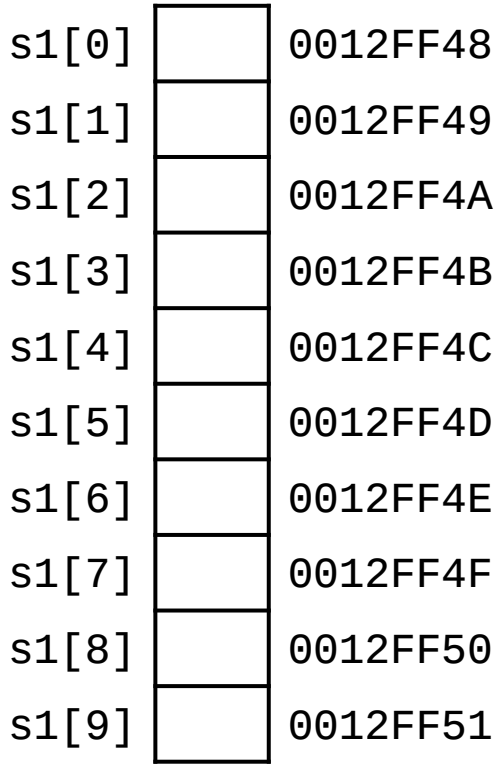
  - Does not work forother array types
  - Characters printed until **`null`** found

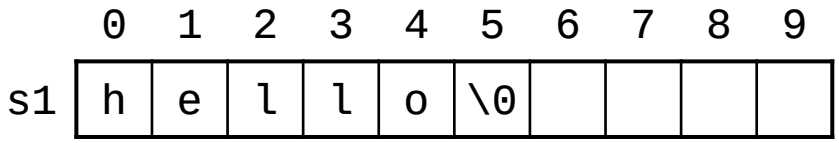
```
int main()
{
   char string1[ 20 ];
   char string2[] = "string literal";

   cout << "Enter the string \"hello there\": ";
   cin >> string1; // reads "hello"

   cout << "string1 is: " << string1 << "\nstring2 is: " << string2;

   cout << "\nstring1 with spaces between characters is:\n";
   for( int i = 0; string1[ i ] != '\0'; i++ )
      cout << string1[ i ] << ' ';

   cin >> string1; // reads "there"
   cout << "\nstring1 is: " << string1 << endl;
}
```

```
Enter the string "hello there": hello there
string1 is: hello
string2 is: string literal
string1 with spaces between characters is:
h e l l o
string1 is: there
```

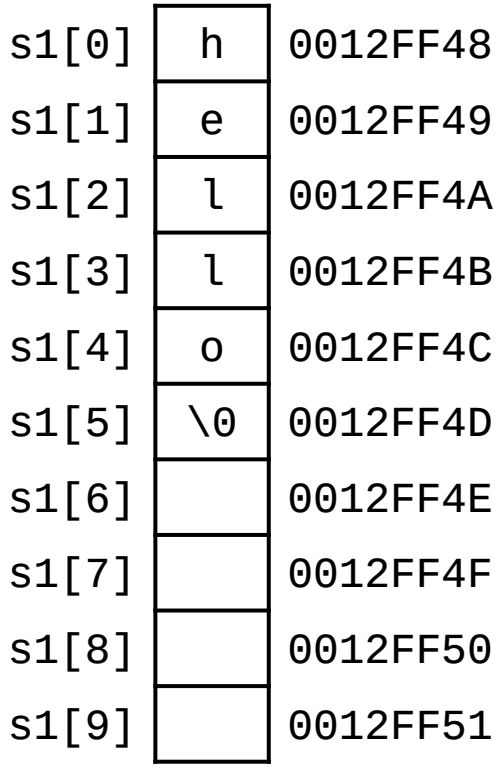


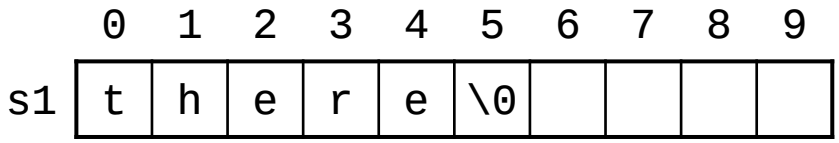

```
int main()
{
   char s1[ 10 ];

   cin >> s1; // reads "hello"

   cout << "string1 is: " << s1;

   for( int i = 0; s1[ i ] != '\0'; i++ )
      cout << s1[ i ] << ' ';

   cin >> s1; // reads "there"
   cout << "\nstring1 is: " << s1 << endl;
}
```



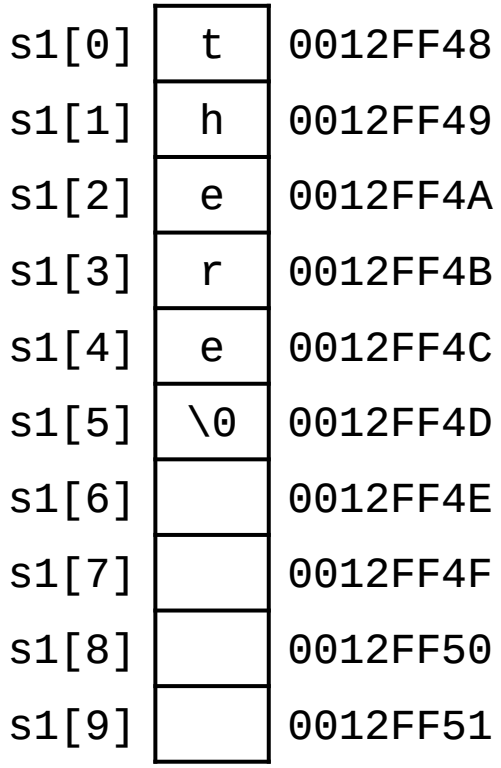


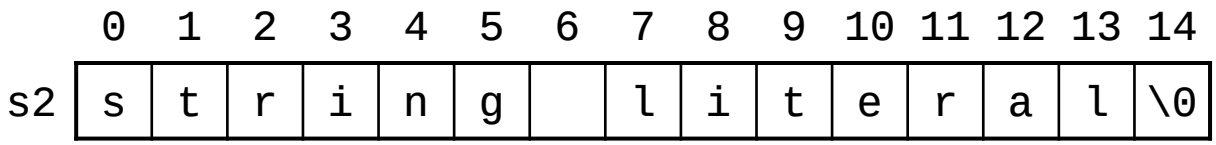

```
int main()
{
   char s1[ 10 ];

   cin >> s1; // reads "hello"

   cout << "string1 is: " << s1;

   for( int i = 0; s1[ i ] != '\0'; i++ )
      cout << s1[ i ] << ' ';

   cin >> s1; // reads "there"
   cout << "\nstring1 is: " << s1 << endl;
}
```



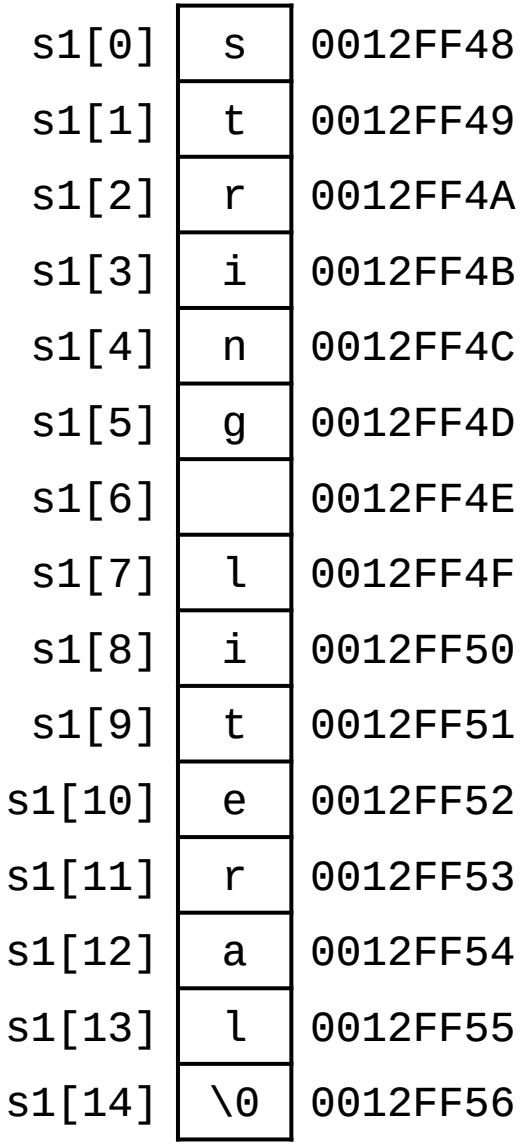



```
int main()
{
   char s1[ 10 ];

   cin >> s1; // reads "hello"

   cout << "string1 is: " << s1;

   for( int i = 0; s1[ i ] != '\0'; i++ )
      cout << s1[ i ] << ' ';

   cin >> s1; // reads "there"
   cout << "\nstring1 is: " << s1 << endl;
}
```





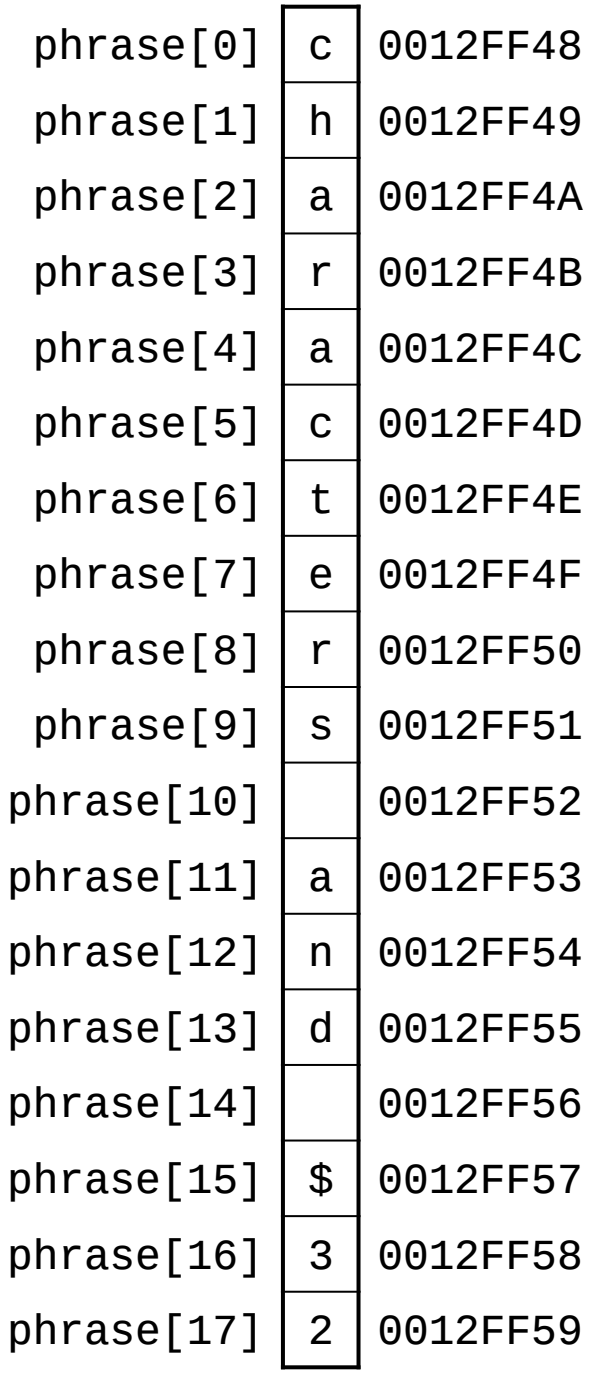

```
int main()
{
   char s2[] = "string literal";

   cout << "string2 is: " << s2;
}
```

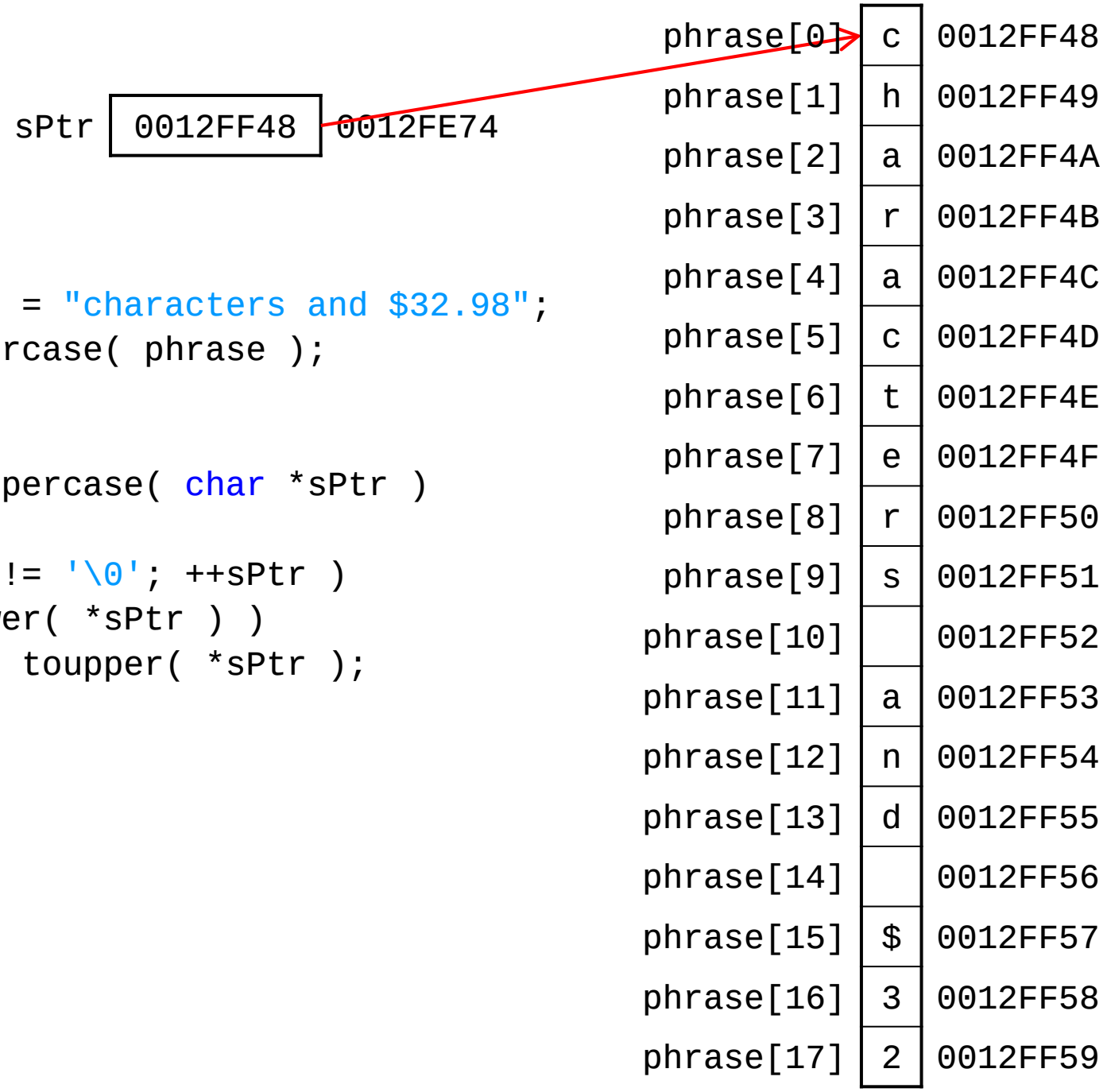


```
#include <iostream>
#include <cctype>
using namespace std;

void convertToUppercase( char * );

int main()
{
   char phrase[] = "characters and $32.98";

   cout << "The phrase before conversion is: " << phrase;
   convertToUppercase( phrase );
   cout << "\nThe phrase after conversion is:  " << phrase << endl;
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; sPtr++ )
      if( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```

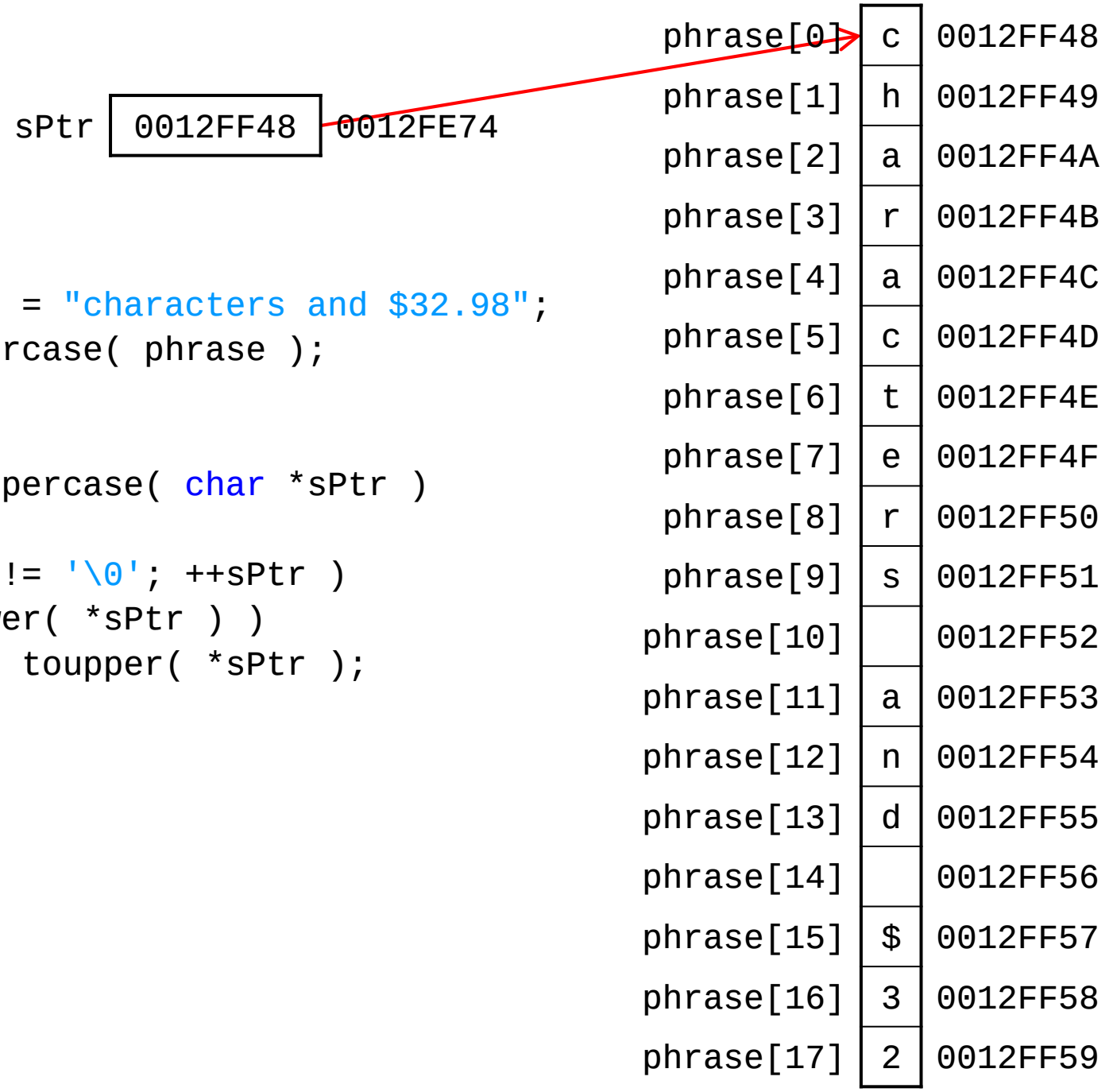
```
The phrase before conversion is: characters and $32.98
The phrase after conversion is:  CHARACTERS AND $32.98
```

```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```

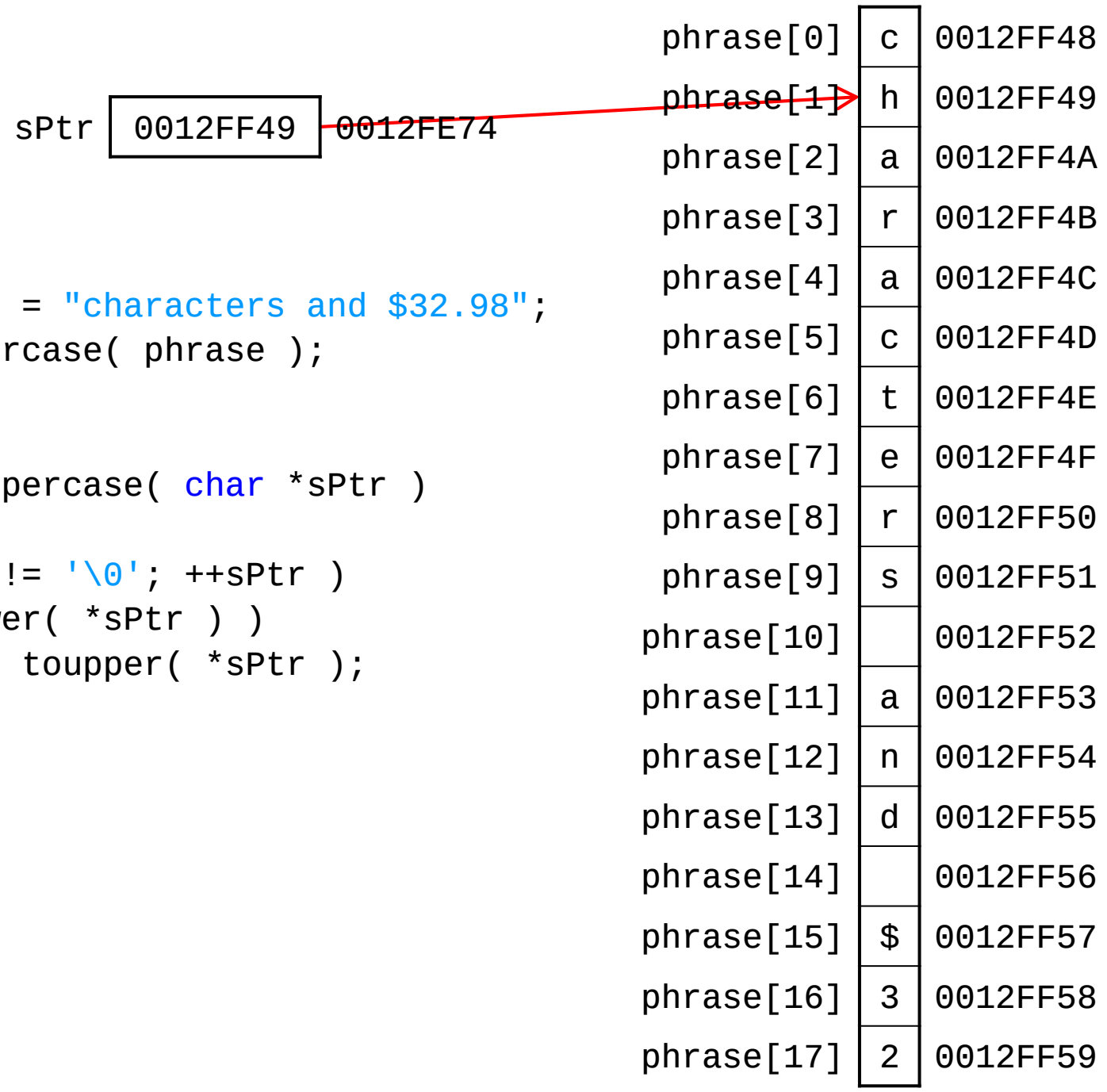




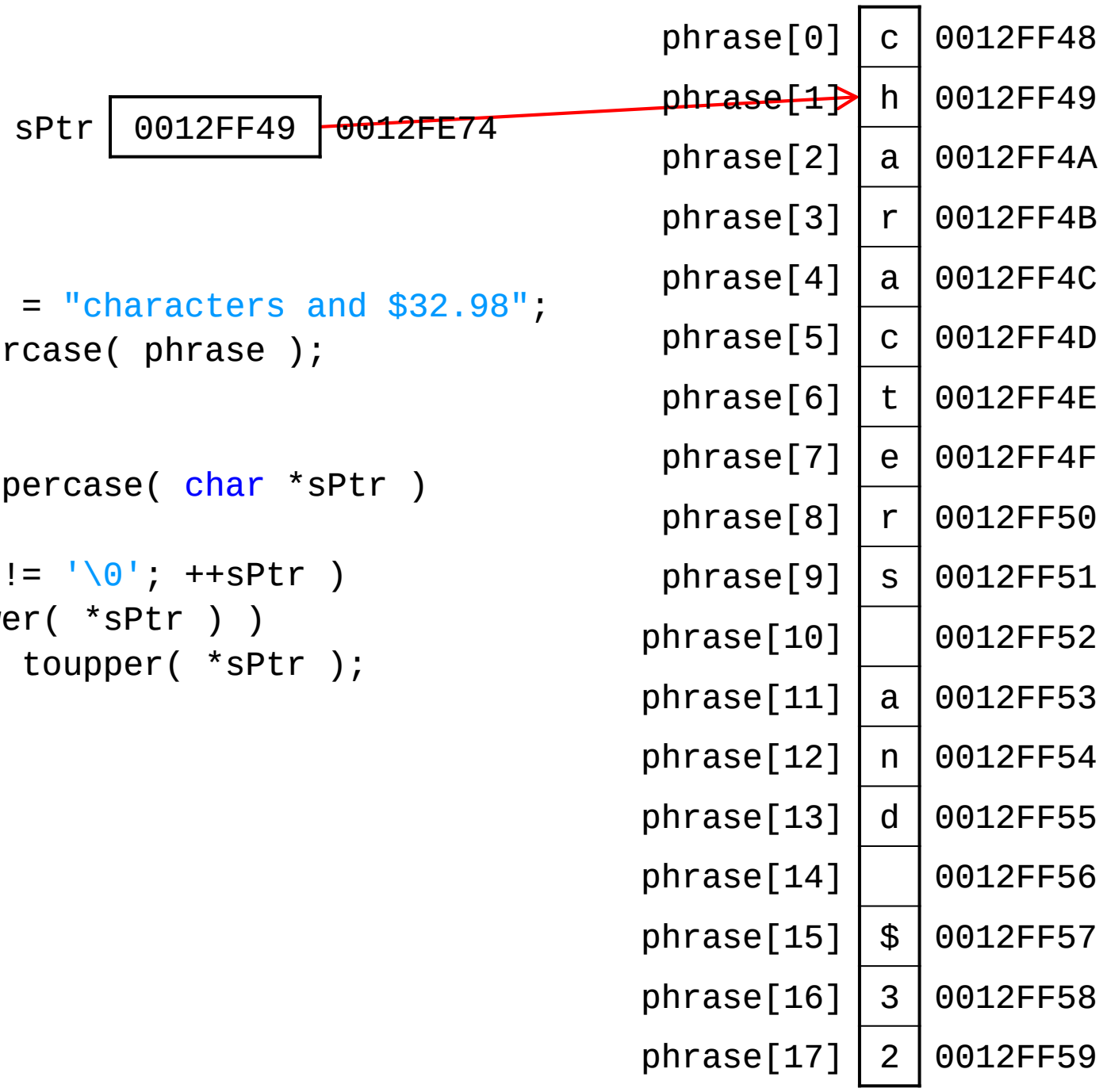

```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```



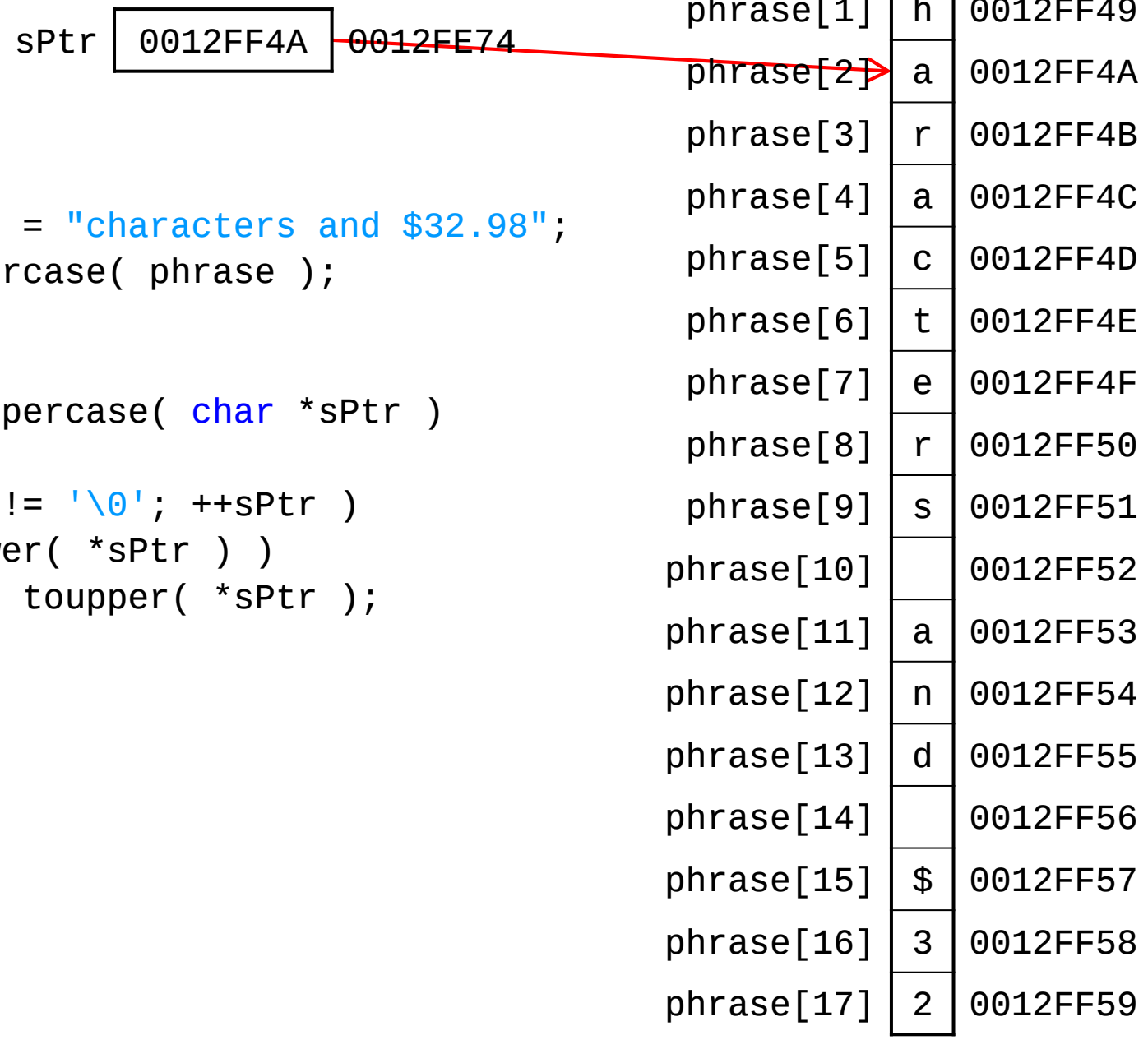

```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```



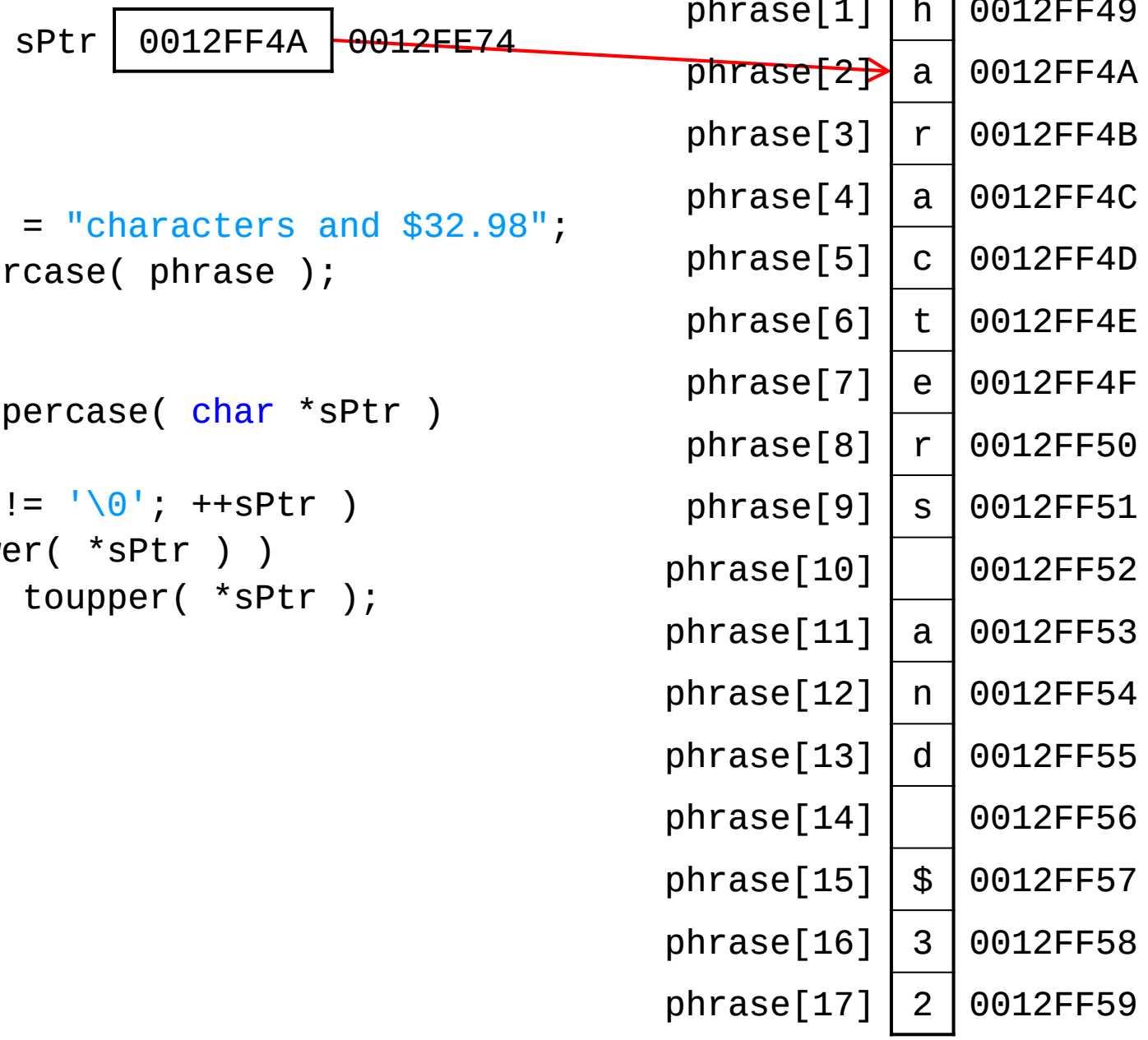

```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```




```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```




```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```




```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```


```
int main()
{
   char phrase[] = "characters and $32.98";
   convertToUppercase( phrase );
}

void convertToUppercase( char *sPtr )
{
   for( ; *sPtr != '\0'; ++sPtr )
      if ( islower( *sPtr ) )
         *sPtr = toupper( *sPtr );
}
```

phrase[0]	c	0012FF48
phrase[1]	h	0012FF49
phrase[2]	a	0012FF4A
phrase[3]	r	0012FF4B
phrase[4]	a	0012FF4C
phrase[5]	c	0012FF4D
phrase[6]	t	0012FF4E
phrase[7]	e	0012FF4F
phrase[8]	r	0012FF50
phrase[9]	s	0012FF51
phrase[10]		0012FF52
phrase[11]	a	0012FF53
phrase[12]	n	0012FF54
phrase[13]	d	0012FF55
phrase[14]		0012FF56
phrase[15]	$	0012FF57
phrase[16]	3	0012FF58
phrase[17]	2	0012FF59

```
int main()
{
   char phrase[] = "characters and $32.98";

   cout << "The phrase before conversion is: " << phrase;
   convertToUppercase( phrase );
   cout << "\nThe phrase after conversion is:  " << phrase << endl;
}
```

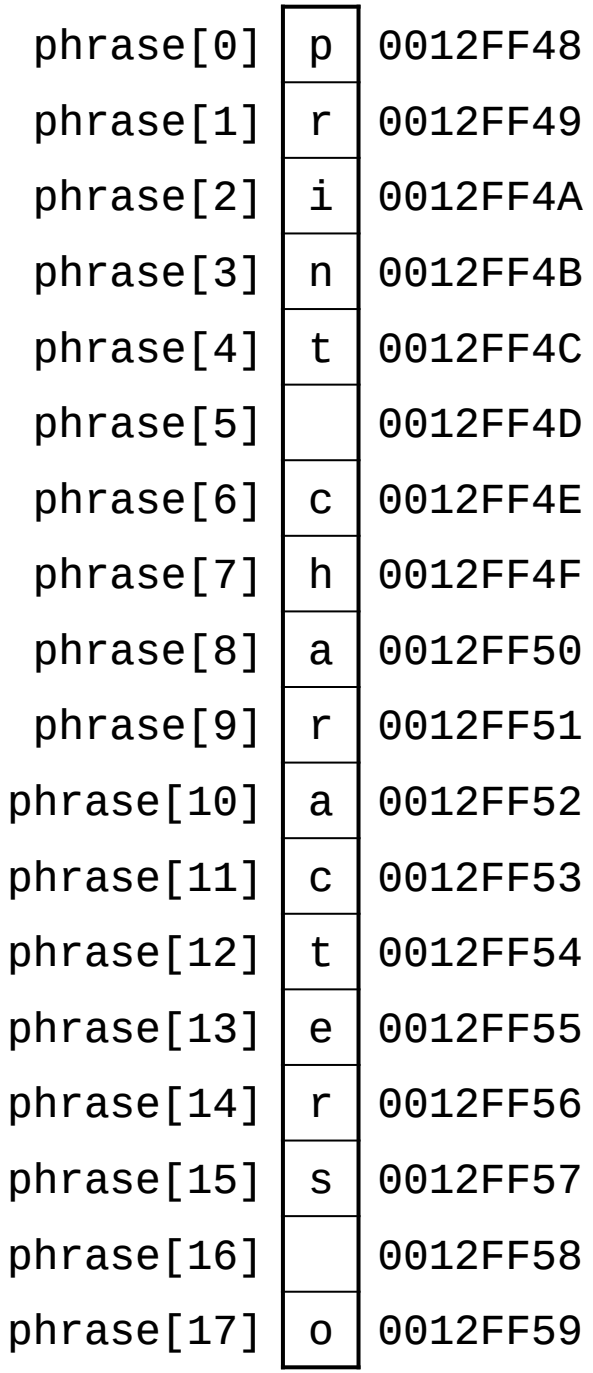
```
int main()
{
   string phrase( "characters and $32.98" );

   cout << "The phrase before conversion is: " << phrase;
   convertToUppercase( phrase );
   cout << "\nThe phrase after conversion is:  " << phrase << endl;
}
```

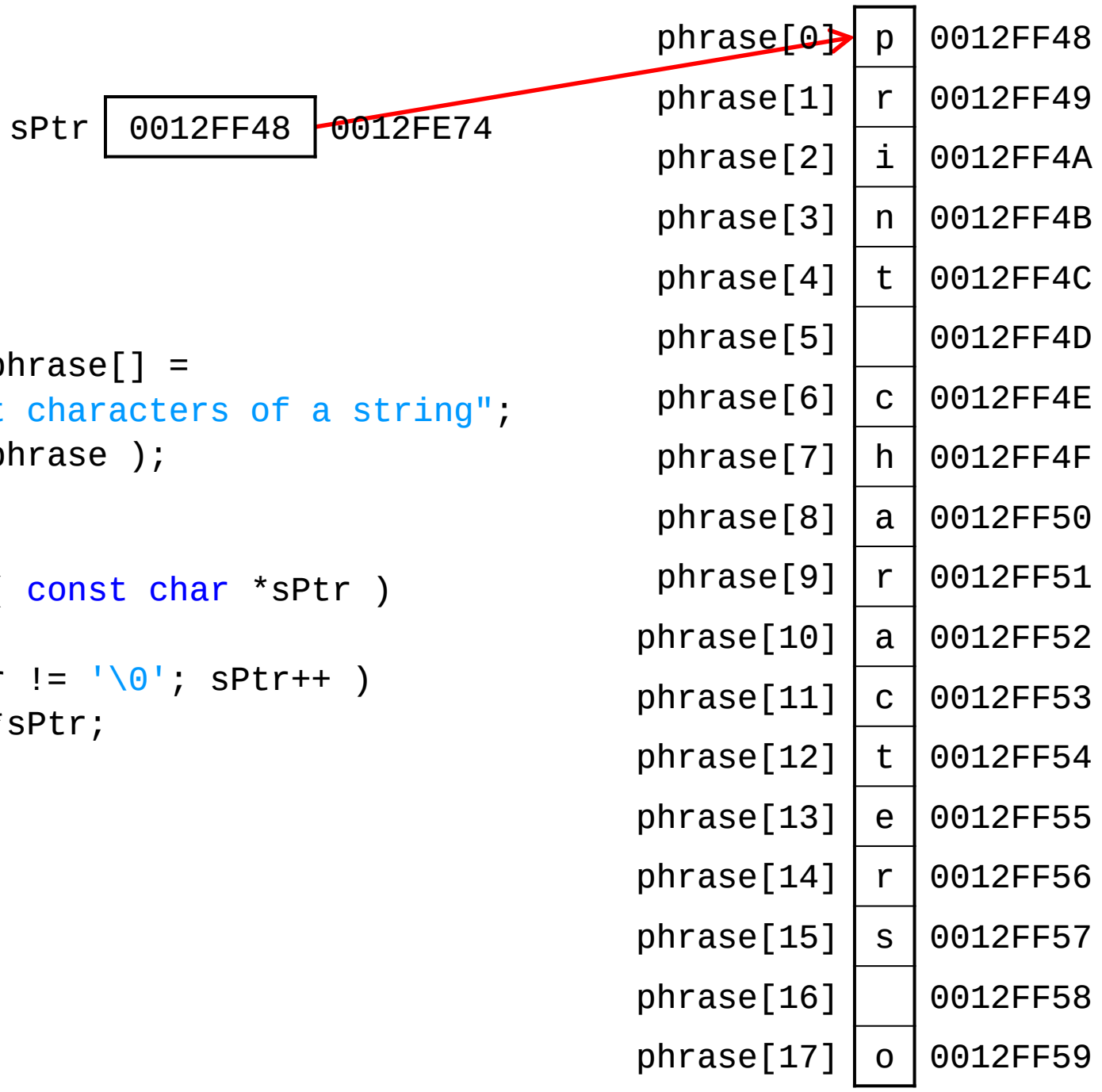
```
void convertToUppercase( char *s )
{
   for( int i = 0; i < strlen( s ); i++ )
      if( islower( s[i] ) )
         s[i] = toupper( s[i] );
}


void convertToUppercase( string s )
{
   for( int i = 0; i < s.length(); i++ )
      if( islower( s[i] ) )
         s[i] = toupper( s[i] );
}
```

```cpp
#include <iostream>
using namespace std;

void printCharacters( const char * );

int main()
{
   const char phrase[] = "print characters of a string";

   cout << "The string is:\n";
   printCharacters( phrase );
   cout << endl;
}

void printCharacters( const char *sPtr )
{
   for( ; *sPtr != '\0'; sPtr++ )
      cout << *sPtr;
}
```

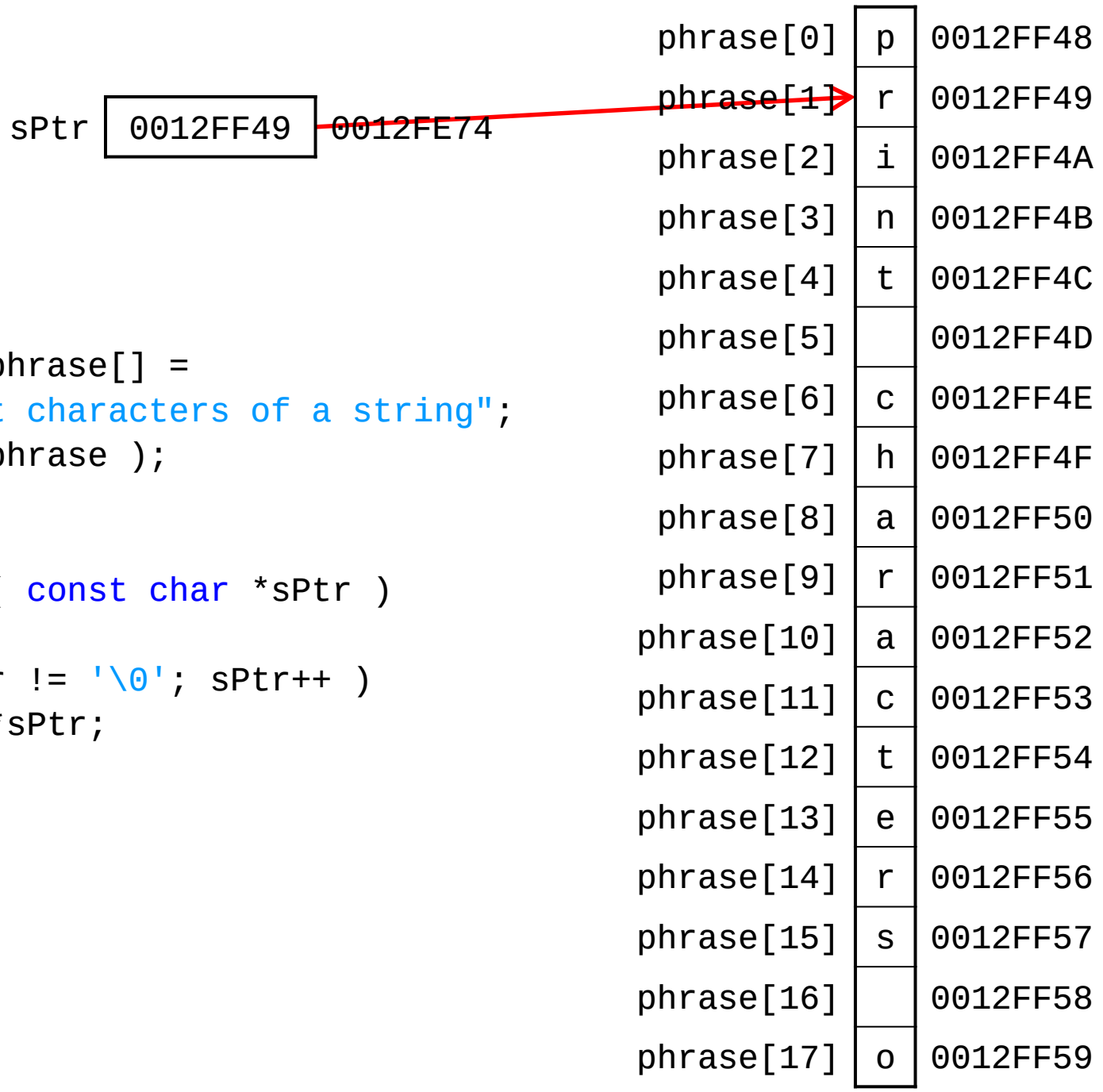
```
The string is:
print characters of a string
```

```
int main()
{
   const char phrase[] =
         "print characters of a string";
   printChar( phrase );
}

void printChar( const char *sPtr )
{
   for( ; *sPtr != '\0'; sPtr++ )
      cout << *sPtr;
}
```

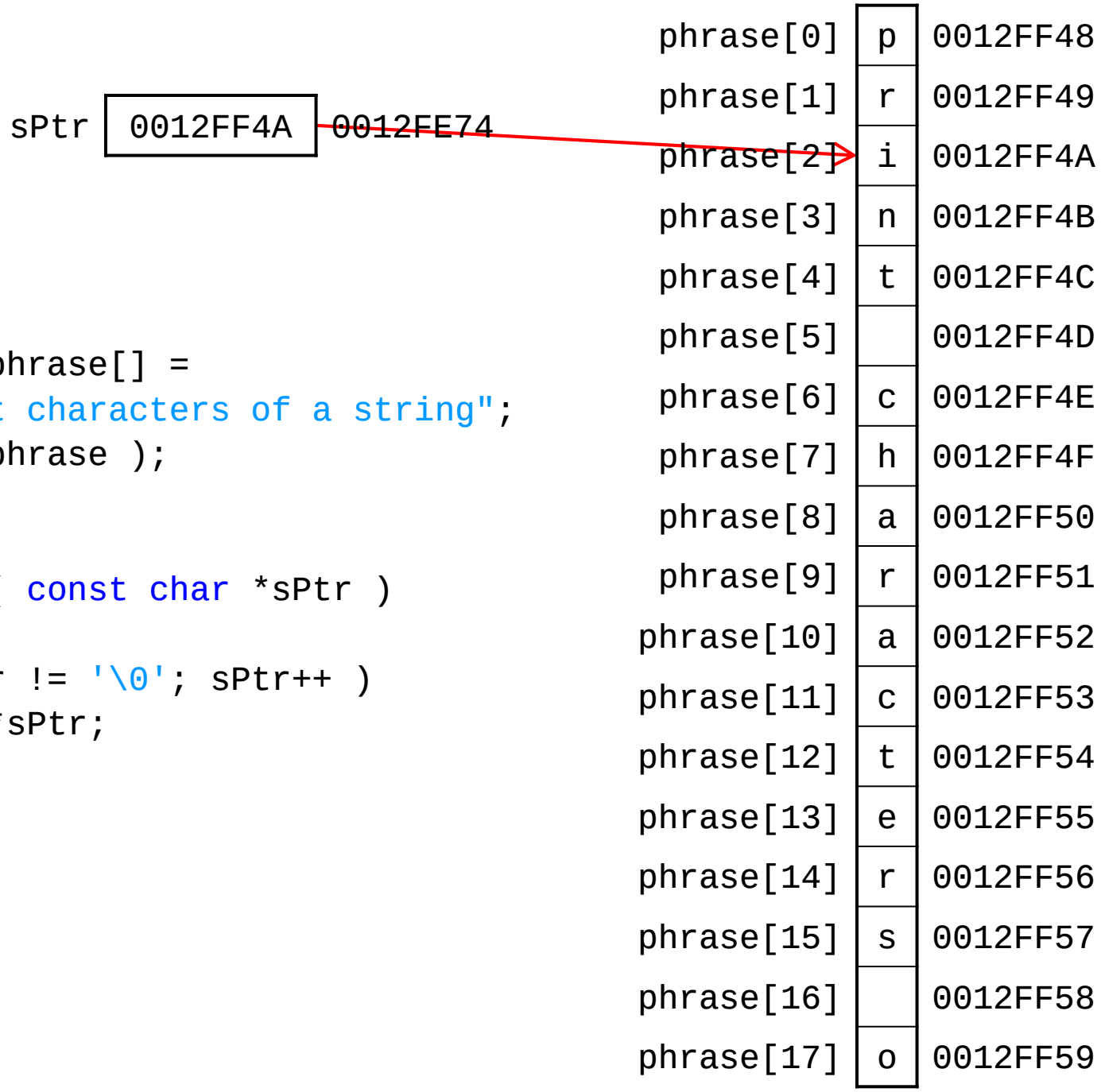




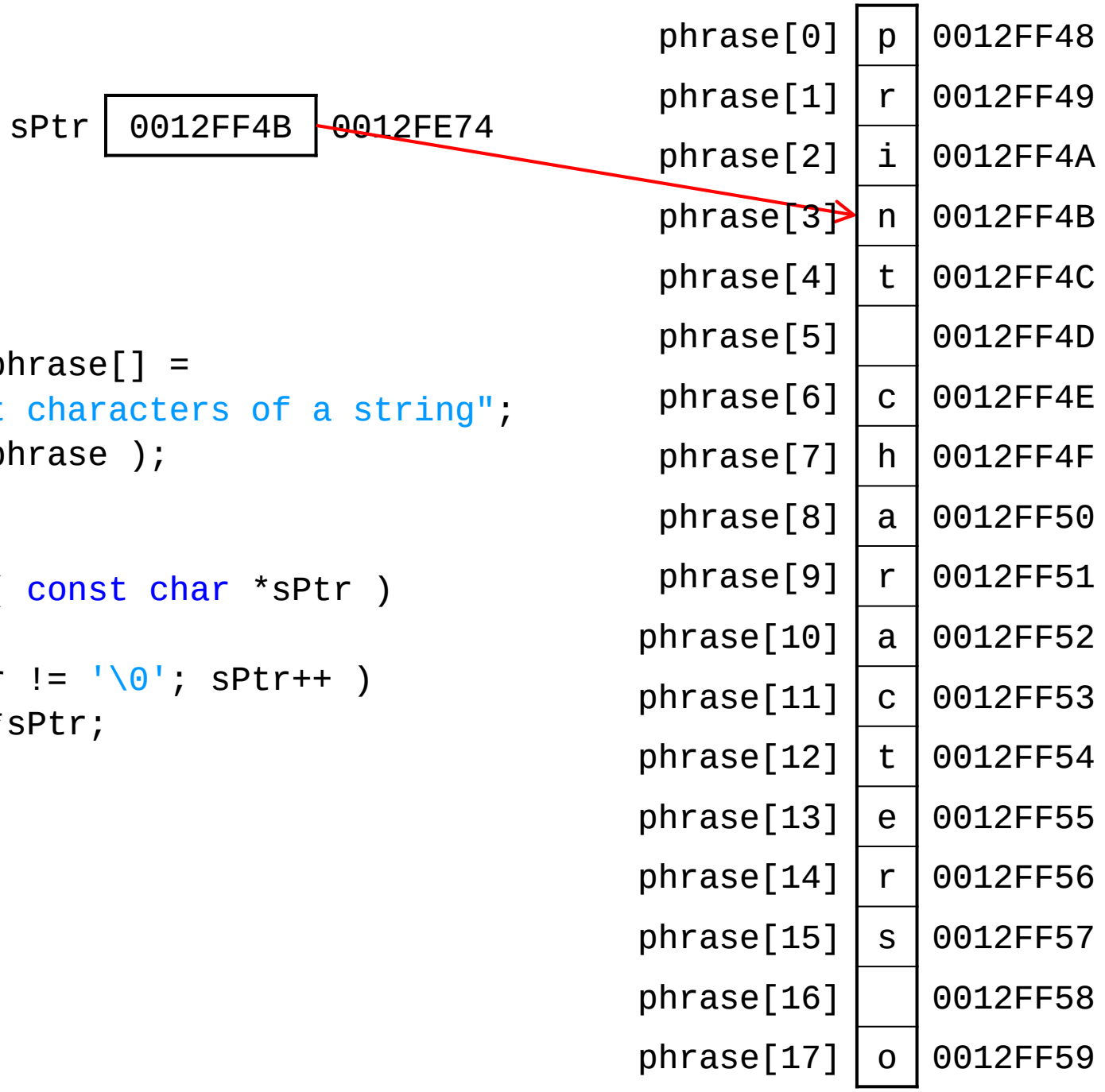

```
int main()
{
   const char phrase[] =
         "print characters of a string";
   printChar( phrase );
}

void printChar( const char *sPtr )
{
   for( ; *sPtr != '\0'; sPtr++ )
      cout << *sPtr;
}
```




```
int main()
{
   const char phrase[] =
         "print characters of a string";
   printChar( phrase );
}

void printChar( const char *sPtr )
{
   for( ; *sPtr != '\0'; sPtr++ )
      cout << *sPtr;
}
```




```
int main()
{
   const char phrase[] =
         "print characters of a string";
   printChar( phrase );
}

void printChar( const char *sPtr )
{
   for( ; *sPtr != '\0'; sPtr++ )
      cout << *sPtr;
}
```




```
int main()
{
   const char phrase[] =
         "print characters of a string";
   printChar( phrase );
}

void printChar( const char *sPtr )
{
   for( ; *sPtr != '\0'; sPtr++ )
      cout << *sPtr;
}
```

```
int main()
{
   const char phrase[] = "print characters of a string";
   printCharacters( phrase );
}

void printCharacters( const char s[] )
{
   for( unsigned int i = 0; i < strlen( s ); i++ )
      cout << s[i];
}
```

```
int main()
{
   const string phrase( "print characters of a string" );
   printCharacters( phrase );
}

void printCharacters( const string s )
{
   for( unsigned int i = 0; i < s.length(); i++ )
      cout << s[i];
}
```

```
#include <iostream>
using namespace std;

void copy1( char *, const char * );
void copy2( char *, const char * );

int main()
{
   char string1[ 10 ];
   char *string2 = "Hello";
   char string3[ 10 ];
   char string4[] = "Good Bye";

   copy1( string1, string2 );
   cout << "string1 = " << string1 << endl;

   copy2( string3, string4 );
   cout << "string3 = " << string3 << endl;
}
```

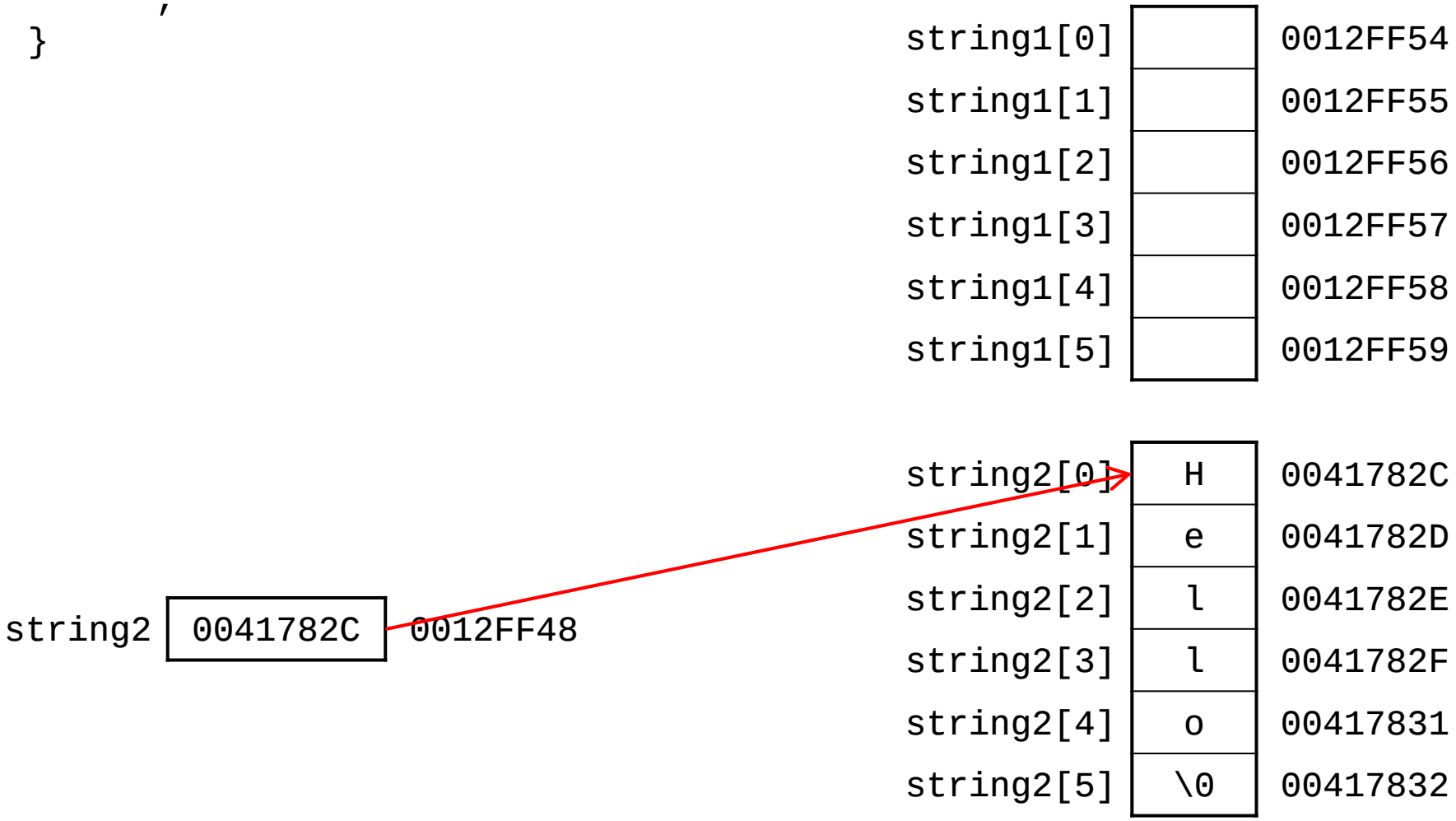
```
void copy1( char * s1, const char * s2 )
{
   for( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
      ;
}

void copy2( char *s1, const char *s2 )
{
   for( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
      ;
}
```

```
void copy1( char * s1, const char * s2 )
{
   for( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
      ;
}

void copy2( char *s1, const char *s2 )
{
   for( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
      ;
}
```

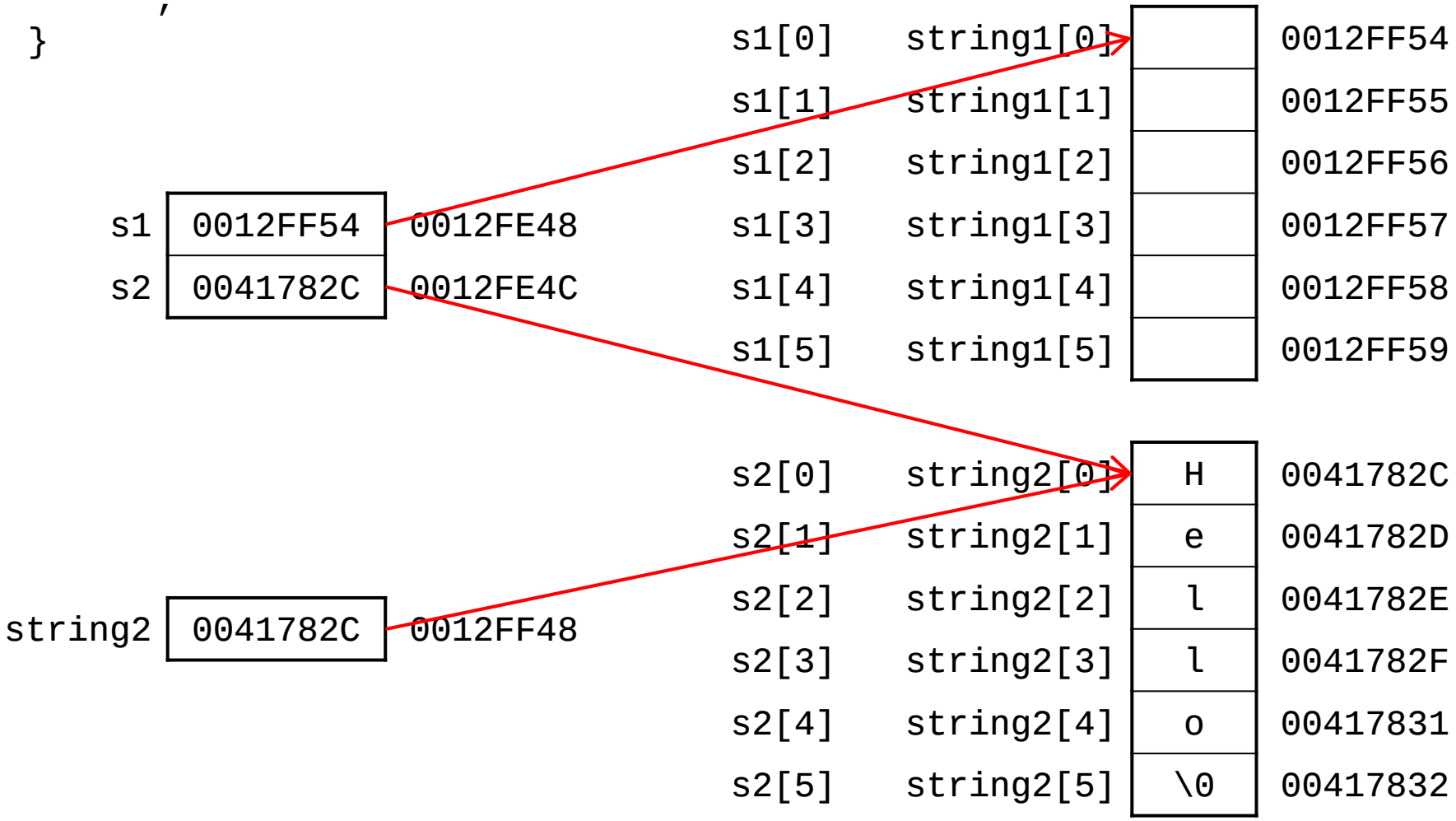
```
string1 = Hello
string3 = Good Bye
```

```
int main()
{
   char string1[ 10 ];
   char *string2 = "Hello";

   copy1( string1, string2 );
   cout << "string1 = " << string1 << endl;
}

void copy1( char * s1, const char * s2 )
{
   for( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
      ;
}
```

```
copy1( string1, string2 );

void copy1( char *s1, const char *s2 )
{
   for( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
      ;
}
```



```
copy1( string1, string2 );

void copy1( char *s1, const char *s2 )
{
   for( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
      ;
}
```

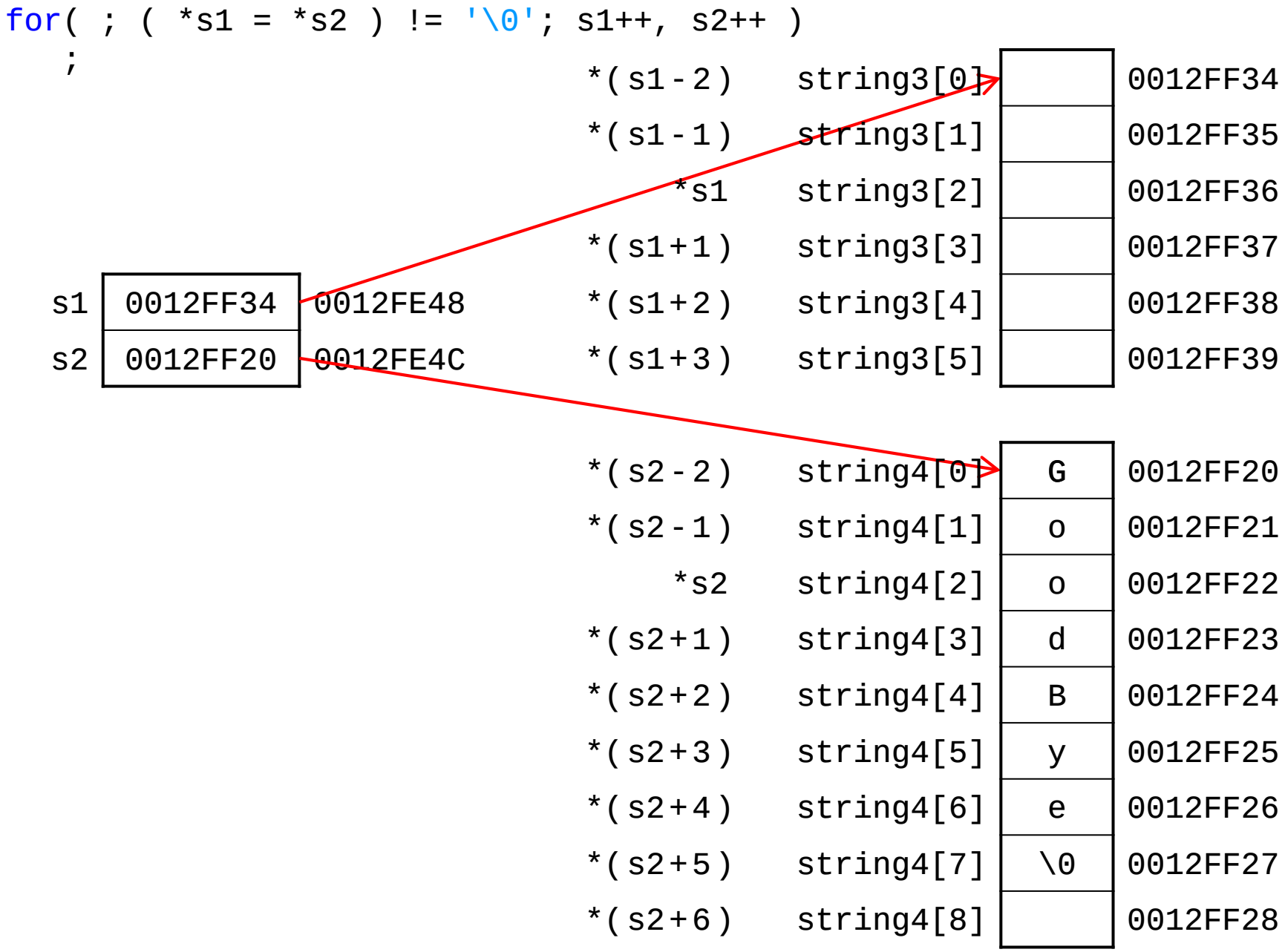


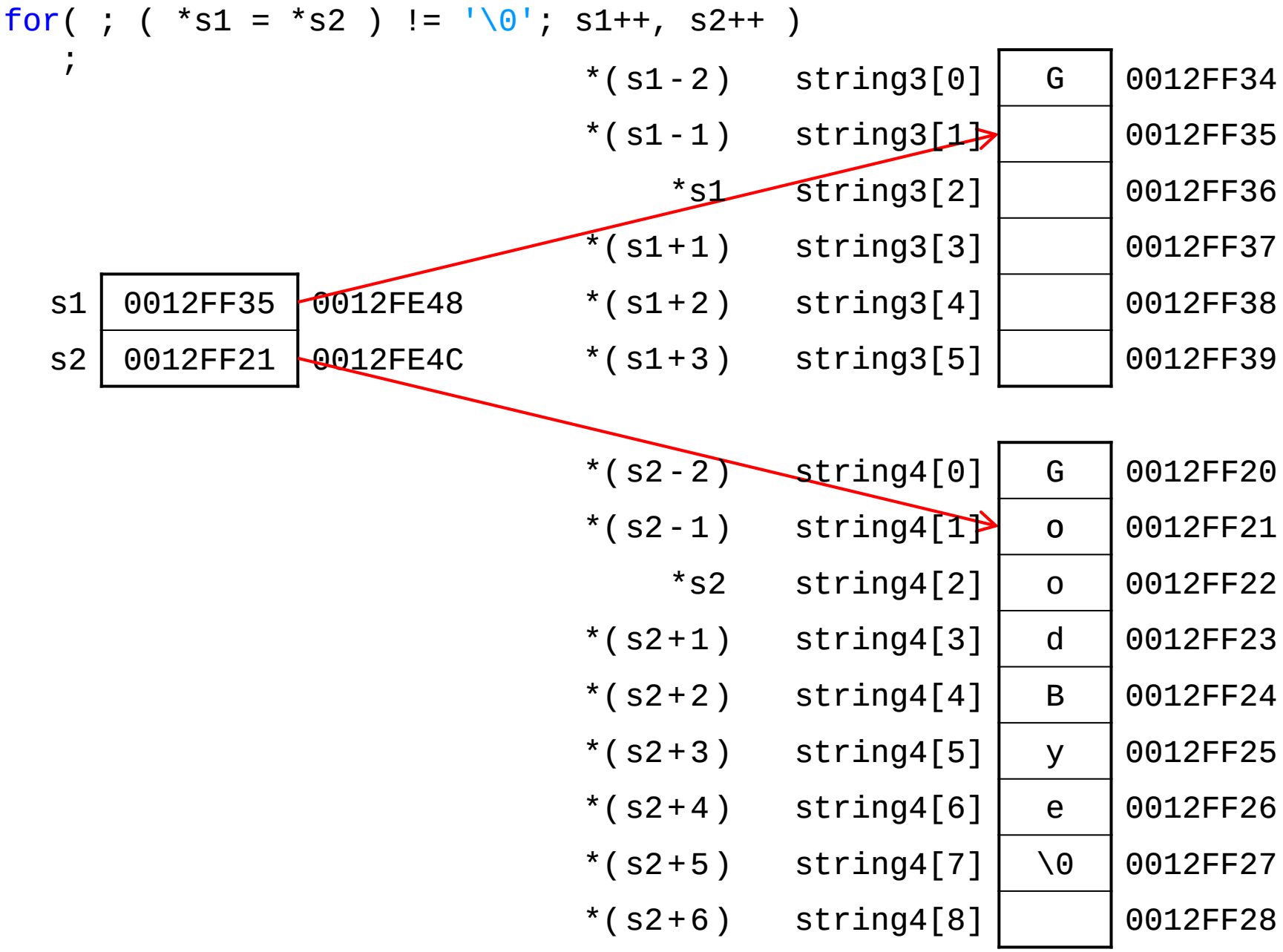
```
int main()
{
   char string3[ 10 ];
   char string4[] = "Good Bye";

   copy2( string3, string4 );
   cout << "string3 = " << string3 << endl;
}

void copy2( char *s1, const char *s2 )
{
   for( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
      ;
}
```

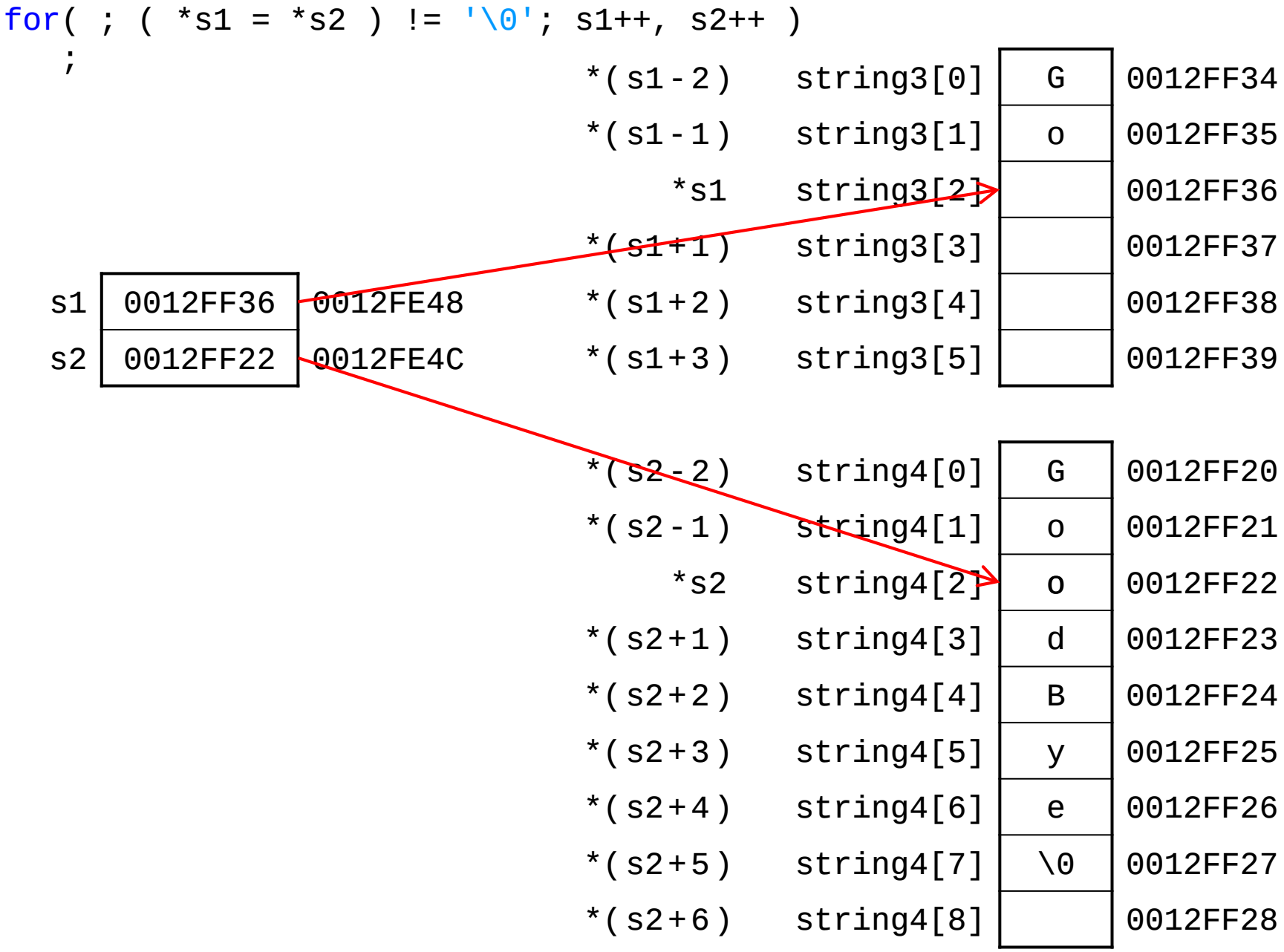
```
for( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
    ;
```

*(s1-2)	string3[0]		0012FF34
*(s1-1)	string3[1]		0012FF35
*s1	string3[2]		0012FF36
*(s1+1)	string3[3]		0012FF37
*(s1+2)	string3[4]		0012FF38
*(s1+3)	string3[5]		0012FF39

*(s2-2)	string4[0]	G	0012FF20
*(s2-1)	string4[1]	o	0012FF21
*s2	string4[2]	o	0012FF22
*(s2+1)	string4[3]	d	0012FF23
*(s2+2)	string4[4]	B	0012FF24
*(s2+3)	string4[5]	y	0012FF25
*(s2+4)	string4[6]	e	0012FF26
*(s2+5)	string4[7]	\0	0012FF27
*(s2+6)	string4[8]		0012FF28


for(; (*s1 = *s2) != '\0'; s1++, s2++)
;
s1
0012FF34
0012FE48
s2
0012FF20
0012FE4C
*(s1-2) string3[0]
0012FF34
*(s1-1) string3[1]
0012FF35
*s1 string3[2]
0012FF36
*(s1+1) string3[3]
0012FF37
*(s1+2) string3[4]
0012FF38
*(s1+3) string3[5]
0012FF39
*(s2-2) string4[0]
G
0012FF20
*(s2-1) string4[1]
o
0012FF21
*s2 string4[2]
o
0012FF22
*(s2+1) string4[3]
d
0012FF23
*(s2+2) string4[4]
B
0012FF24
*(s2+3) string4[5]
y
0012FF25
*(s2+4) string4[6]
e
0012FF26
*(s2+5) string4[7]
\0
0012FF27
*(s2+6) string4[8]
0012FF28


for(; (*s1 = *s2) != '\0'; s1++, s2++)
;
s1
0012FF35
0012FE48
s2
0012FF21
0012FE4C
*(s1-2) string3[0] G 0012FF34
*(s1-1) string3[1] 0012FF35
*s1 string3[2] 0012FF36
*(s1+1) string3[3] 0012FF37
*(s1+2) string3[4] 0012FF38
*(s1+3) string3[5] 0012FF39
*(s2-2) string4[0] G 0012FF20
*(s2-1) string4[1] o 0012FF21
*s2 string4[2] o 0012FF22
*(s2+1) string4[3] d 0012FF23
*(s2+2) string4[4] B 0012FF24
*(s2+3) string4[5] y 0012FF25
*(s2+4) string4[6] e 0012FF26
*(s2+5) string4[7] \0 0012FF27
*(s2+6) string4[8] 0012FF28


for(; (*s1 = *s2) != '\0'; s1++, s2++)
;
s1
0012FF36
0012FE48
s2
0012FF22
0012FE4C
*(s1-2)
string3[0]
G
0012FF34
*(s1-1)
string3[1]
o
0012FF35
*s1
string3[2]
0012FF36
*(s1+1)
string3[3]
0012FF37
*(s1+2)
string3[4]
0012FF38
*(s1+3)
string3[5]
0012FF39
*(s2-2)
string4[0]
G
0012FF20
*(s2-1)
string4[1]
o
0012FF21
*s2
string4[2]
o
0012FF22
*(s2+1)
string4[3]
d
0012FF23
*(s2+2)
string4[4]
B
0012FF24
*(s2+3)
string4[5]
y
0012FF25
*(s2+4)
string4[6]
e
0012FF26
*(s2+5)
string4[7]
\0
0012FF27
*(s2+6)
string4[8]
0012FF28

Pointer-Based String Manipulation Functions

```
char *strcpy( char *s1, const char *s2 );
```

Copies the string `s2` into the character array `s1`. The value of `s1` is returned.

```
char *strncpy( char *s1, const char *s2, size_t n );
```

Copies at most `n` characters of the string `s2` into the character array `s1`. The value of `s1` is returned.

```
char *strcat( char *s1, const char *s2 );
```

Appends the string `s2` to `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.

```
char *strncat( char *s1, const char *s2, size_t n );
```

Appends at most `n` characters of string `s2` to string `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.

```
int strcmp( const char *s1, const char *s2 );
```

Compares the string `s1` with the string `s2`. The function returns a value of zero, less than zero (usually –1) or greater than zero (usually 1) if `s1` is equal to, less than or greater than `s2`, respectively.

`int strncmp( const char *s1, const char *s2, size_t n );`

Compares up to `n` characters of the string `s1` with the string `s2`. The function returns zero, less than zero or greater than zero if the `n`-character portion of `s1` is equal to, less than or greater than the corresponding `n`-character portion of `s2`, respectively.

`char *strtok( char *s1, const char *s2 );`

A sequence of calls to `strtok` breaks string `s1` into “tokens” — logical pieces such as words in a line of text. The string is broken up based on the characters contained in string `s2`. forinstance, if we were to break the string `"this:is:a:string"` into tokens based on the character `':'`, the resulting tokens would be `"this"`, `"is"`, `"a"` and `"string"`. Function strtok returns only one token at a time, however. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

`size_t strlen( const char *s );`

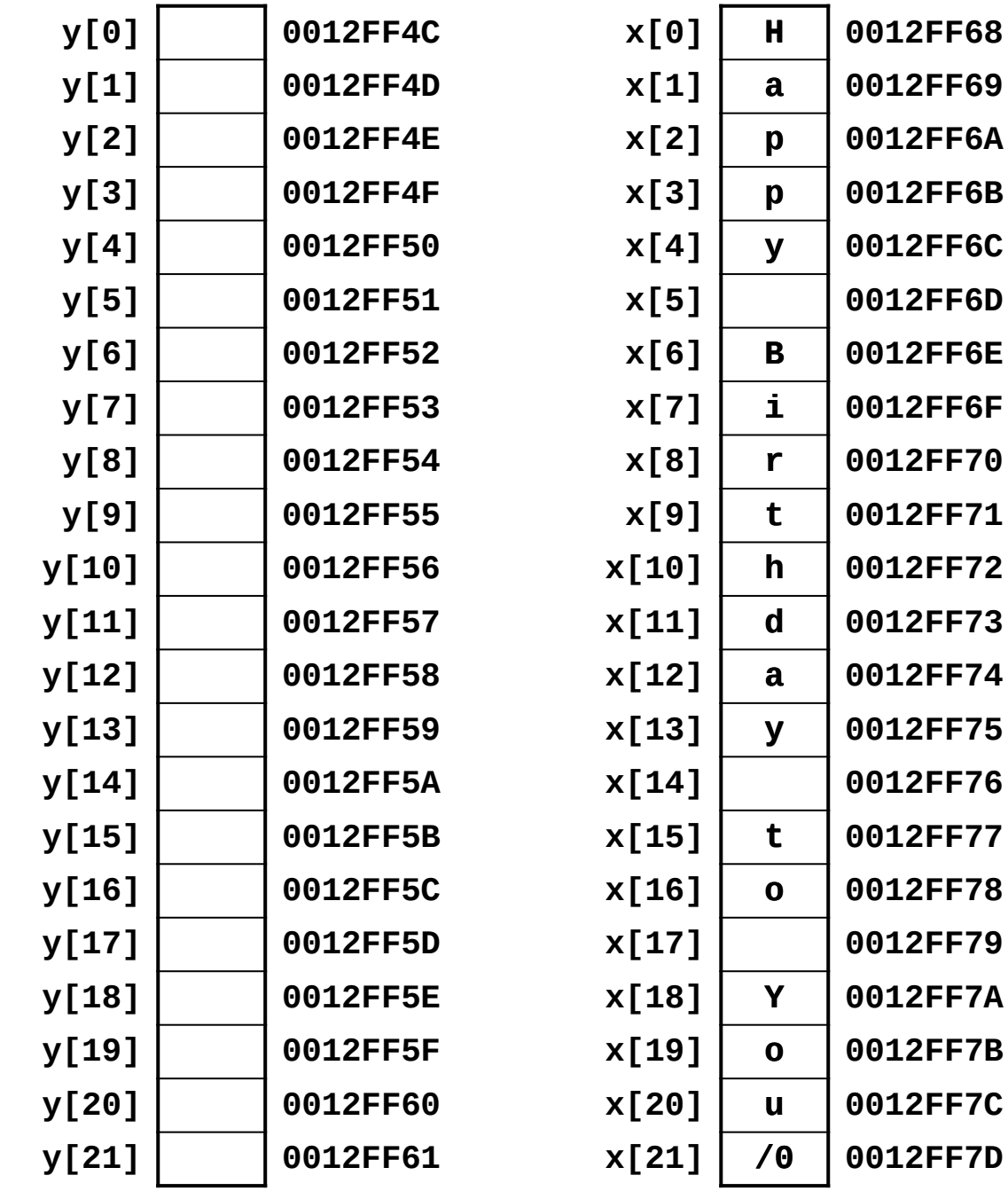
Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

Pointer-Based String Manipulation Functions

- Copying strings
  - `char *strcpy( char *s1, const char *s2 )`
    - Copies second argument into first argument
      - First argument must be large enough to store string and terminating null character
  - `char *strncpy( char *s1, const char *s2, size_t n )`
    - Specifies number of characters to be copied from string into array
    - Does not necessarily copy terminating null character

```cpp
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   char x[] = "Happy Birthday to You";
   char y[ 25 ];
   char z[ 15 ];

   strcpy( y, x );

   cout << "The string in array x is: " << x
      << "\nThe string in array y is: " << y << '\n';

   strncpy( z, x, 14 ); // copy first 14 characters of x into z
   z[ 14 ] = '\0'; // append '\0' to z's contents

   cout << "The string in array z is: " << z << endl;
}
```

```
strcpy( y, x );
```


y[0] 0012FF4C
y[1] 0012FF4D
y[2] 0012FF4E
y[3] 0012FF4F
y[4] 0012FF50
y[5] 0012FF51
y[6] 0012FF52
y[7] 0012FF53
y[8] 0012FF54
y[9] 0012FF55
y[10] 0012FF56
y[11] 0012FF57
y[12] 0012FF58
y[13] 0012FF59
y[14] 0012FF5A
y[15] 0012FF5B
y[16] 0012FF5C
y[17] 0012FF5D
y[18] 0012FF5E
y[19] 0012FF5F
y[20] 0012FF60
y[21] 0012FF61
x[0] H 0012FF68
x[1] a 0012FF69
x[2] p 0012FF6A
x[3] p 0012FF6B
x[4] y 0012FF6C
x[5] 0012FF6D
x[6] B 0012FF6E
x[7] i 0012FF6F
x[8] r 0012FF70
x[9] t 0012FF71
x[10] h 0012FF72
x[11] d 0012FF73
x[12] a 0012FF74
x[13] y 0012FF75
x[14] 0012FF76
x[15] t 0012FF77
x[16] o 0012FF78
x[17] 0012FF79
x[18] Y 0012FF7A
x[19] o 0012FF7B
x[20] u 0012FF7C
x[21] /0 0012FF7D

```
int main()
{
   char x[] = "Happy Birthday to You"; // string length 21
   char y[ 25 ];
   char z[ 15 ];

   strcpy( y, x ); // copy contents of x into y

   cout << "The string in array x is: " << x
      << "\nThe string in array y is: " << y << '\n';

   // copy first 14 characters of x into z
   strncpy( z, x, 14 ); // does not copy null character
   z[ 14 ] = '\0'; // append '\0' to z's contents

   cout << "The string in array z is: " << z << endl;
}
```

```
int main()
{
   char x[] = "Happy Birthday to You"; // string length 21
   char y[ 25 ];
   char z[ 15 ];

   strcpy_s( y, 25, x ) // copy contents of x into y

   cout << "The string in array x is: " << x
      << "\nThe string in array y is: " << y << '\n';

   // copy first 14 characters of x into z
   strncpy_s( z, 15, x, 14 ); // places '\0' after last character

   cout << "The string in array z is: " << z << endl;
}
```

```
int main()
{
   string x( "Happy Birthday to You" ); // string length 21
   string y;
   string z;

   y = x; // copy contents of x into y

   cout << "The string in array x is: " << x
      << "\nThe string in array y is: " << y << '\n';

   // copy first 14 characters of x into z
   z = x.substr( 0, 14 );

   cout << "The string in array z is: " << z << endl;
}
```

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

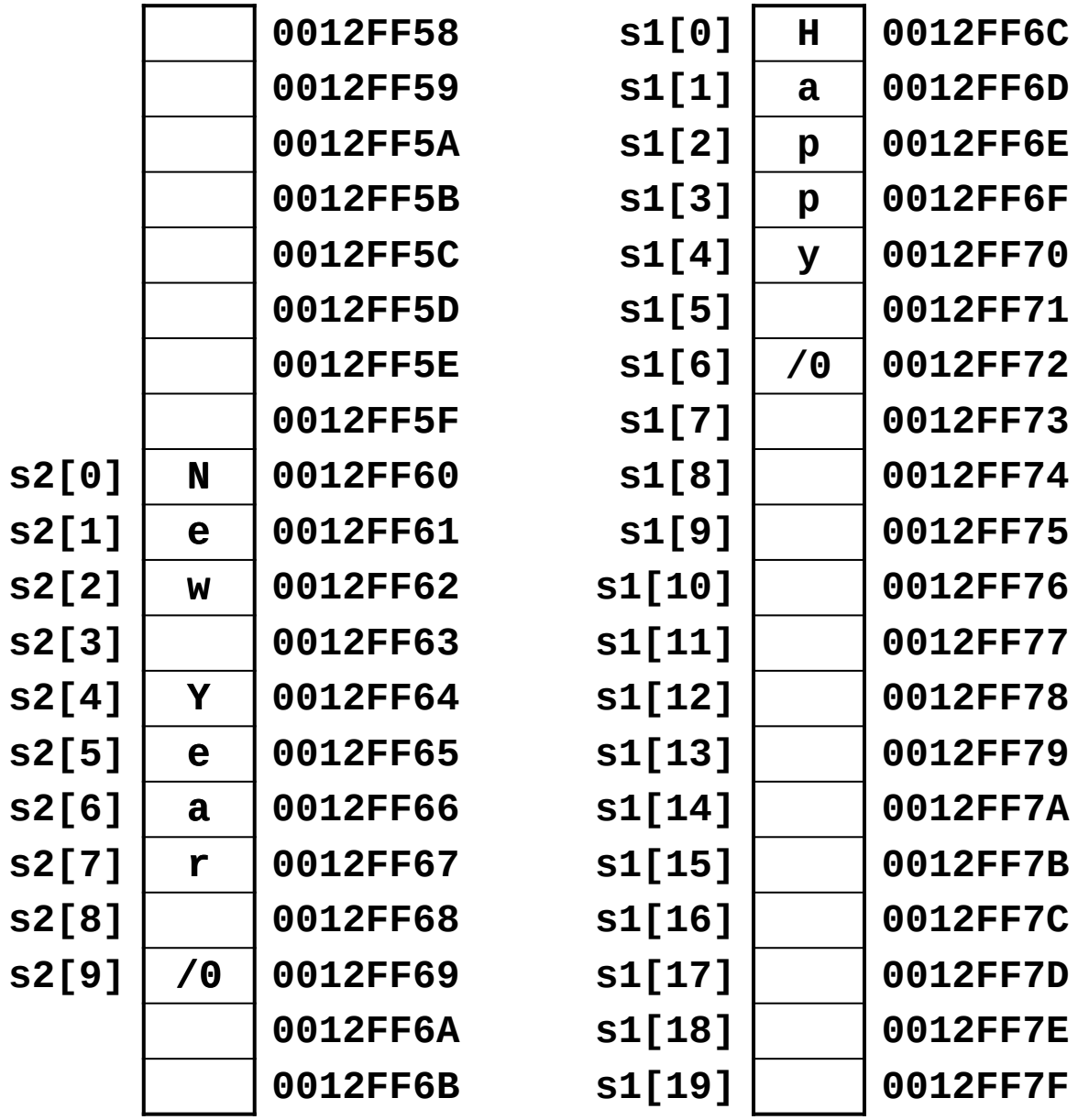
Pointer-Based String Manipulation Functions

- Concatenating strings
  - `char *strcat( char *s1, const char *s2 )`
    - Appends second argument to first argument
    - First character of second argument replaces null character terminating first argument
    - Ensure first argument large enough to store concatenated result and null character
  - `char *strncat( char *s1, const char *s2, size_t n )`
    - Appends specified number of characters from second argument to first argument
    - Appends terminating null character to result

```
int main()
{
   char s1[ 20 ] = "Happy ";
   char s2[] = "New Year ";
   char s3[ 40 ] = "";

   cout << "s1 = " << s1 << "\ns2 = " << s2;

   strcat( s1, s2 );

   cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1 << "\ns2 = " << s2;

   // concatenate first 6 characters of s1 to s3
   strncat( s3, s1, 6 ); // places '\0' after last character

   cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
        << "\ns3 = " << s3;

   strcat( s3, s1 );
   cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1
      << "\ns3 = " << s3 << endl;
}
```

```
s1 = Happy
s2 = New Year

After strcat(s1, s2):
s1 = Happy New Year
s2 = New Year

After strncat(s3, s1, 6):
s1 = Happy New Year
s3 = Happy

After strcat(s3, s1):
s1 = Happy New Year
s3 = Happy Happy New Year
```

```
strcat( s1, s2 );
```


s2[0] N 0012FF60
s2[1] e 0012FF61
s2[2] w 0012FF62
s2[3] 0012FF63
s2[4] Y 0012FF64
s2[5] e 0012FF65
s2[6] a 0012FF66
s2[7] r 0012FF67
s2[8] 0012FF68
s2[9] /0 0012FF69
0012FF58
0012FF59
0012FF5A
0012FF5B
0012FF5C
0012FF5D
0012FF5E
0012FF5F
0012FF6A
0012FF6B
s1[0] H 0012FF6C
s1[1] a 0012FF6D
s1[2] p 0012FF6E
s1[3] p 0012FF6F
s1[4] y 0012FF70
s1[5] 0012FF71
s1[6] /0 0012FF72
s1[7] 0012FF73
s1[8] 0012FF74
s1[9] 0012FF75
s1[10] 0012FF76
s1[11] 0012FF77
s1[12] 0012FF78
s1[13] 0012FF79
s1[14] 0012FF7A
s1[15] 0012FF7B
s1[16] 0012FF7C
s1[17] 0012FF7D
s1[18] 0012FF7E
s1[19] 0012FF7F

```
int main()
{
   char s1[ 20 ] = "Happy ";
   char s2[] = "New Year ";
   char s3[ 40 ] = "";

   cout << "s1 = " << s1 << "\ns2 = " << s2;

   strcat( s1, s2 ); // concatenate s2 to s1 (length 15)

   cout << s1 << "\ns2 = " << s2;

   strncat( s3, s1, 6 ); // places '\0' after last character

   cout << s1 << "\ns3 = " << s3;

   strcat( s3, s1 ); // concatenate s1 to s3
   cout << s1 << "\ns3 = " << s3 << endl;
}
```

```
int main()
{
   char s1[ 20 ] = "Happy ";
   char s2[] = "New Year ";
   char s3[ 40 ] = "";

   cout << "s1 = " << s1 << "\ns2 = " << s2;

   strcat_s( s1, 20, s2 ); // concatenate s2 to s1 (length 15)

   cout << s1 << "\ns2 = " << s2;

   strncat_s( s3, 40, s1, 6 ); // places '\0' after last character

   cout << s1 << "\ns3 = " << s3;

   strcat_s( s3, 40, s1 ); // concatenate s1 to s3
   cout << s1 << "\ns3 = " << s3 << endl;
}
```

```
int main()
{
   string s1( "Happy " );
   string s2( "New Year " );
   string s3;

   cout << "s1 = " << s1 << "\ns2 = " << s2;

   s1 += s2; // concatenate s2 to s1 (length 15)

   cout << s1 << "\ns2 = " << s2;

   s3 += s1.substr( 0, 6 );

   cout << s1 << "\ns3 = " << s3;

   s3 += s1; // concatenate s1 to s3
   cout << s1 << "\ns3 = " << s3 << endl;
}
```

Pointer-Based String Manipulation Functions

- Comparing strings
  - Characters represented as numeric codes
    - Strings compared using numeric codes
  - ASCII character sets
    - “American Standard Code forInformation Interchage”

Pointer-Based String Manipulation Functions

- Comparing strings
  - `int strcmp( const char *s1, const char *s2 )`
    - Compares character by character
    - Returns
      - Zero if strings equal
      - Negative value if first string less than second string
      - Positive value if first string greater than second string
  - `int strncmp( const char *s1, const char *s2, size_t n )`
    - Compares up to specified number of characters
    - Stops comparing if reaches null character in one of arguments

```
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;

int main()
{
   char *s1 = "Happy New Year";
   char *s2 = "Happy New Year";
   char *s3 = "Happy Holidays";

   cout << "s1 = " << s1 << "\ns2 = " << s2 << "\ns3 = " << s3
      << "\n\nstrcmp(s1, s2) = " << setw( 2 ) << strcmp( s1, s2 )
      << "\nstrcmp(s1, s3) = " << setw( 2 ) << strcmp( s1, s3 )
      << "\nstrcmp(s3, s1) = " << setw( 2 ) << strcmp( s3, s1 );

   cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
      << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = " << setw( 2 )
      << strncmp( s1, s3, 7 ) << "\nstrncmp(s3, s1, 7) = " << setw( 2 )
      << strncmp( s3, s1, 7 ) << endl;
}
```

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

Pointer-Based String Manipulation Functions

- Determining string lengths
  - `size_t strlen( const char *s )`
    - Returns number of characters in string
      - Terminating null character not included in length

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   char *string1 = "abcdefghijklmnopqrstuvwxyz";
   char *string2 = "four";
   char *string3 = "Boston";

   cout << "The length of \"" << string1 << "\" is " << strlen( string1 )
      << "\nThe length of \"" << string2 << "\" is " << strlen( string2 )
      << "\nThe length of \"" << string3 << "\" is " << strlen( string3 )
      << endl;
}
```

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

```
int main()
{
   char *string1 = "abcdefghijklmnopqrstuvwxyz";
   char *string2 = "four";
   char *string3 = "Boston";

   cout << "The length of \"" << string1 << "\" " << strlen( string1 )
      << "\nThe length of \"" << string2 << "\" " << strlen( string2 )
      << "\nThe length of \"" << string3 << "\" " << strlen( string3 );
}
```

```
int main()
{
   string string1( "abcdefghijklmnopqrstuvwxyz" );
   string string2( "four" );
   string string3( "Boston" );

   cout << "The length of \"" << string1 << "\" " << string1.length()
      << "\nThe length of \"" << string2 << "\" " << string2.length()
      << "\nThe length of \"" << string3 << "\" " << string3.length();
}
```

Pointer-Based String Manipulation Functions

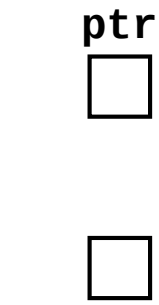
- Tokenizing
  - Breaking strings into tokens, separated by delimiting characters
  - Tokens usually logical units, such as words (separated by spaces)
  - `"This is my string"` has 4 word tokens (separated by spaces)
  - `char *strtok( char *s1, const char *s2 )`
    - Multiple calls required
      - First call contains two arguments, string to be tokenized and string containing delimiting characters
        - Finds next delimiting character and replaces with null character
      - Subsequent calls continue tokenizing
        - Call with first argument `NULL`

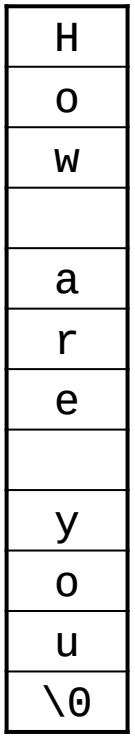
```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   {
      char string[] = "Programming is learned by writing programs";

      cout << string << endl << endl;

      char *token = strtok( string, " " ); // get the first token

      while( token != nullptr )
      {
         cout << token << endl;
         token = strtok( nullptr, " " ); // get next token
      }
      cout << endl;

      cout << string << endl << endl;
   }
}
```

```cpp
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   {
      char string[] = "Programming is learned by writing programs";

      cout << string << endl << endl;

      char *nextToken = nullptr;
      char *token = strtok_s( string, " ", &nextToken );

      while( token != nullptr )
      {
         cout << token << endl;
         token = strtok_s( nullptr, " ", &nextToken );
      }
      cout << endl;
      cout << string << endl << endl;
   }
}
```

```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

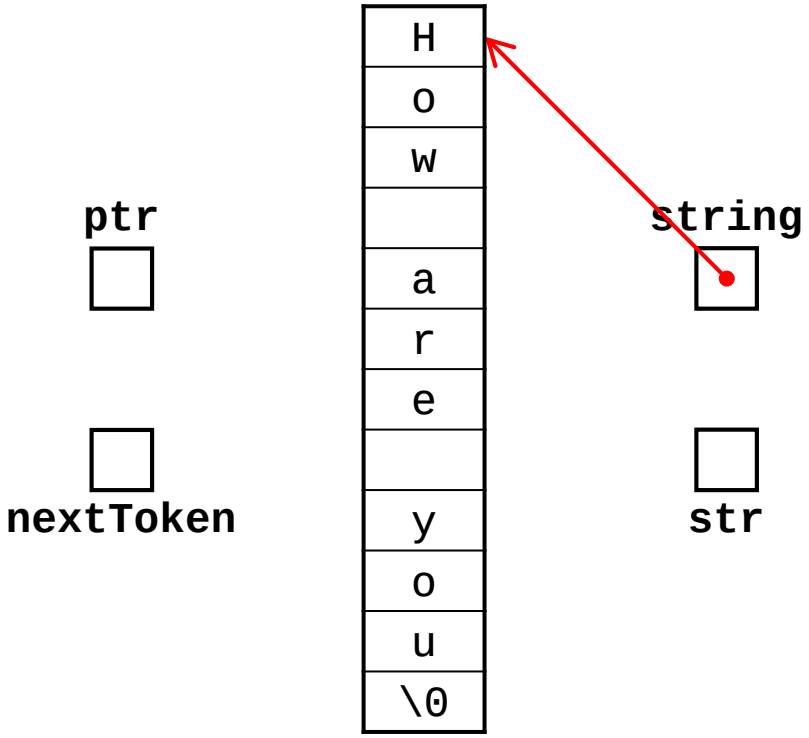
Output





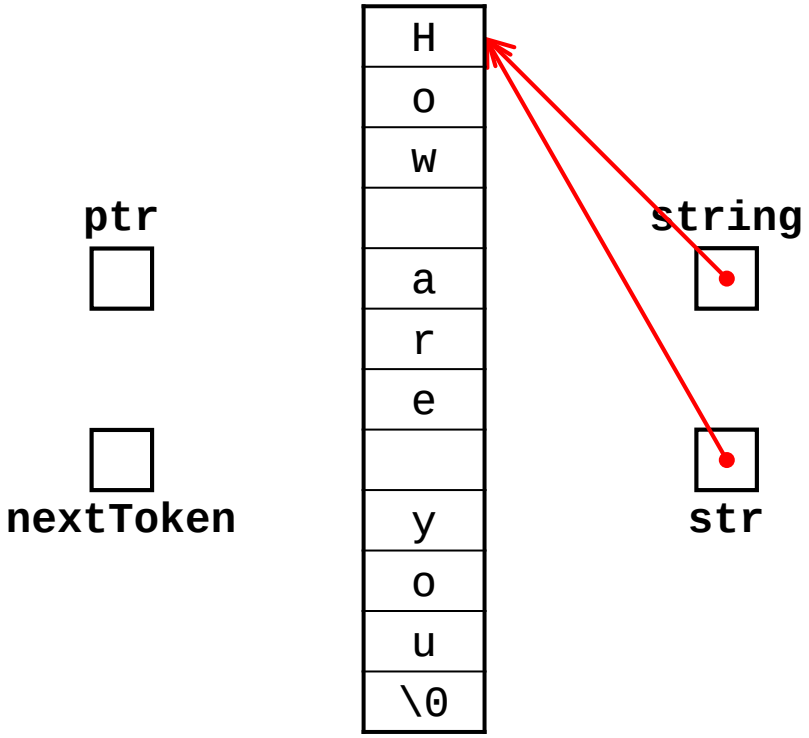
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



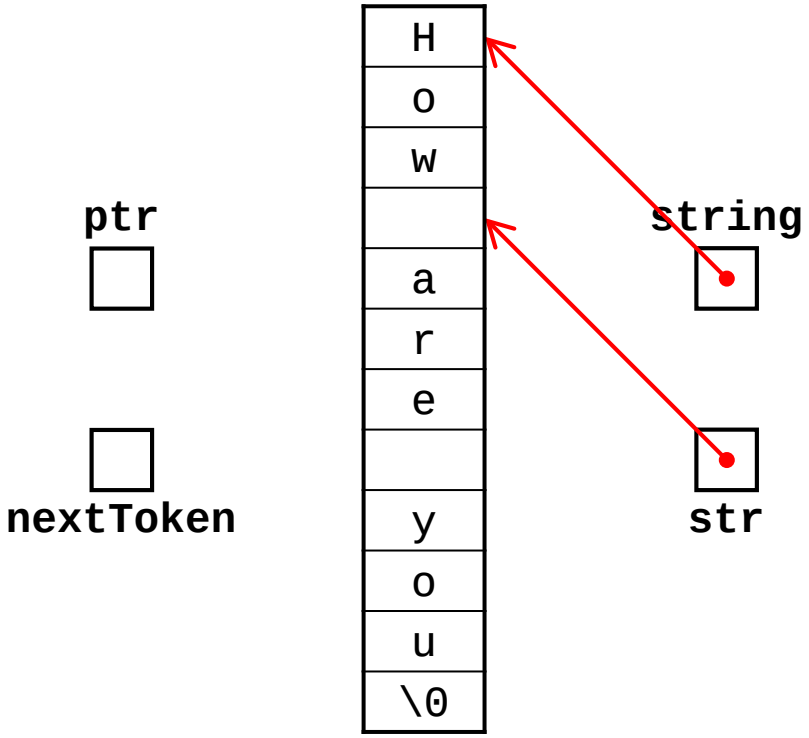
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



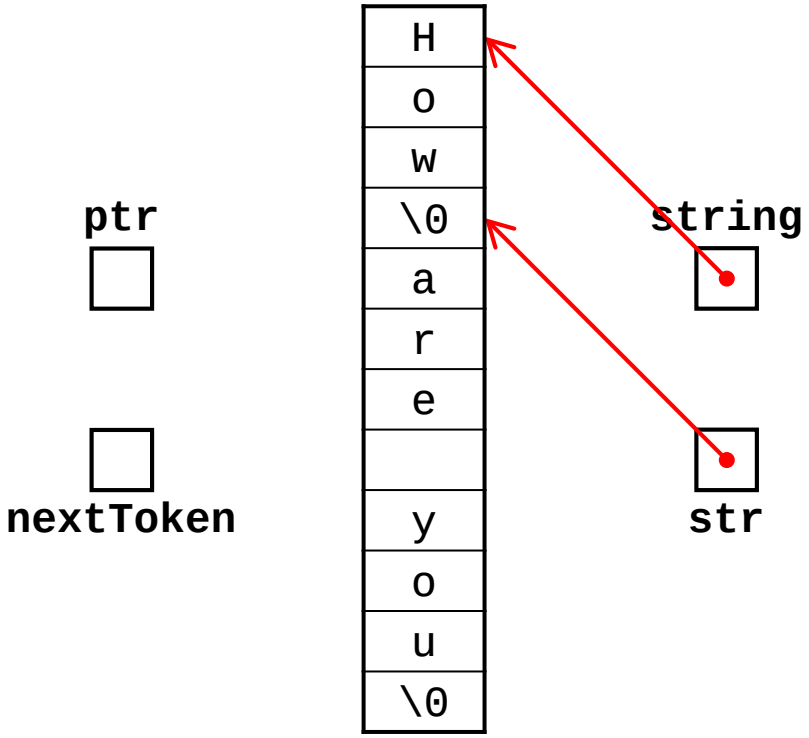
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



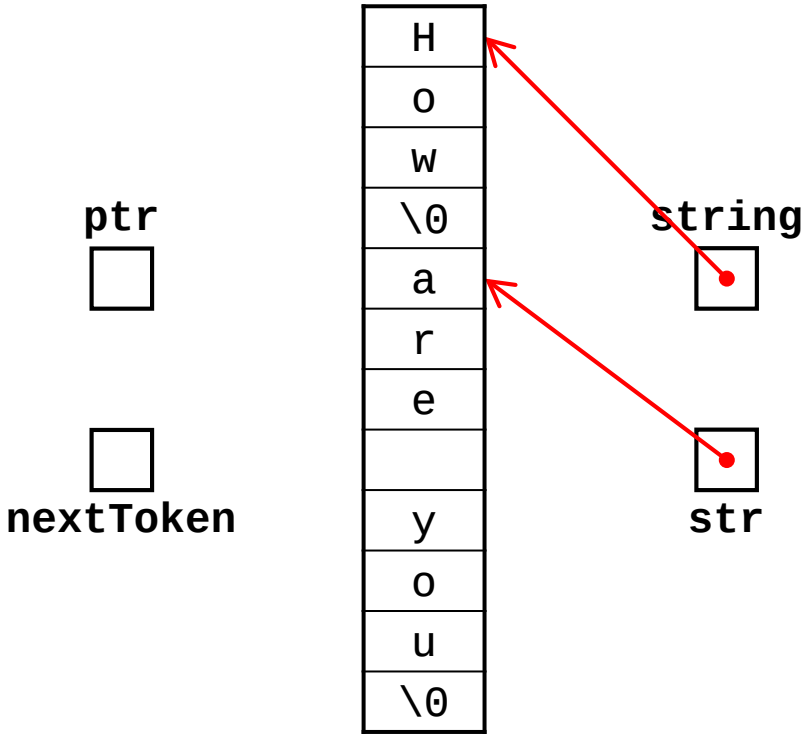
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



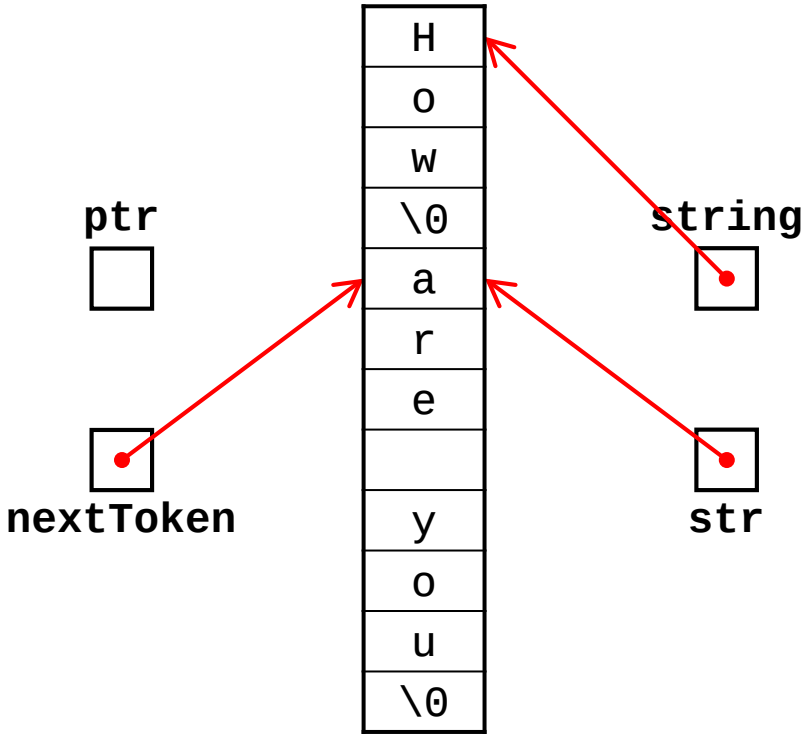
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



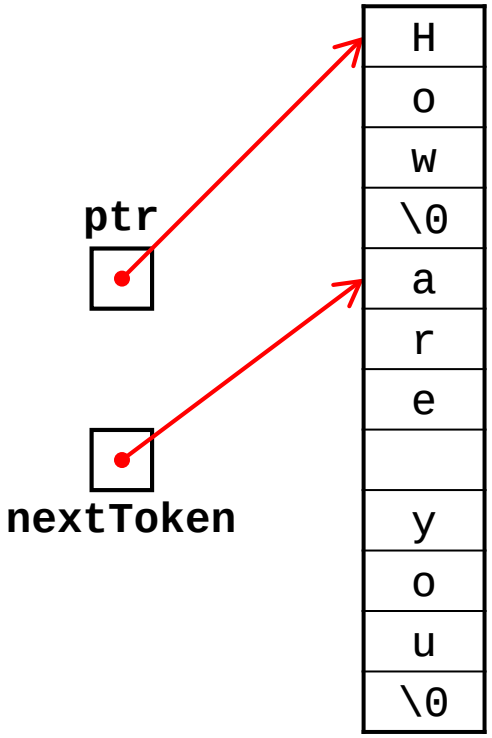
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



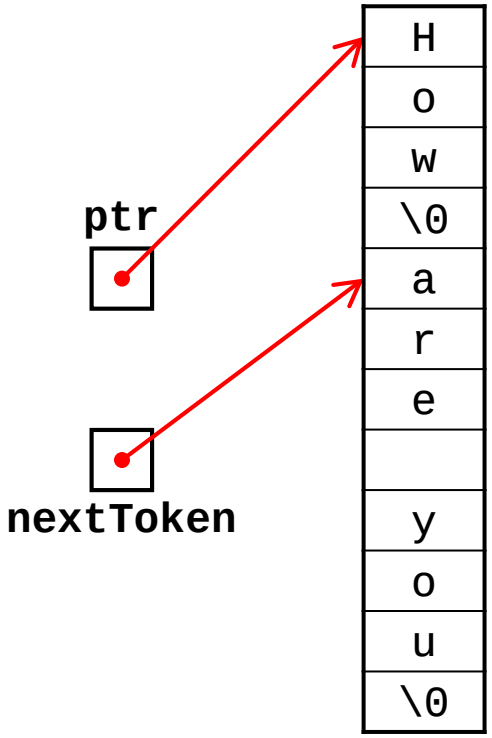
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



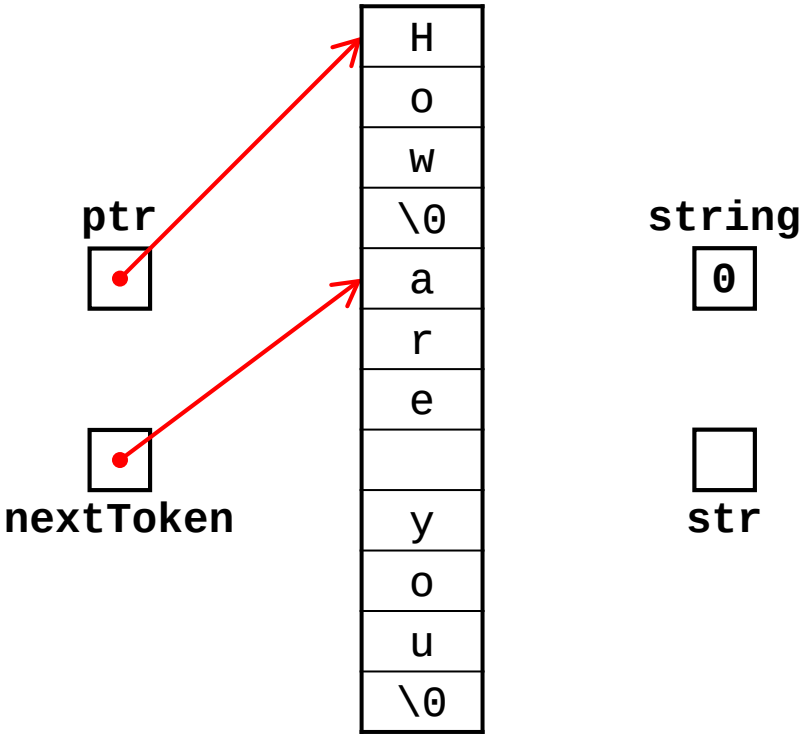
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



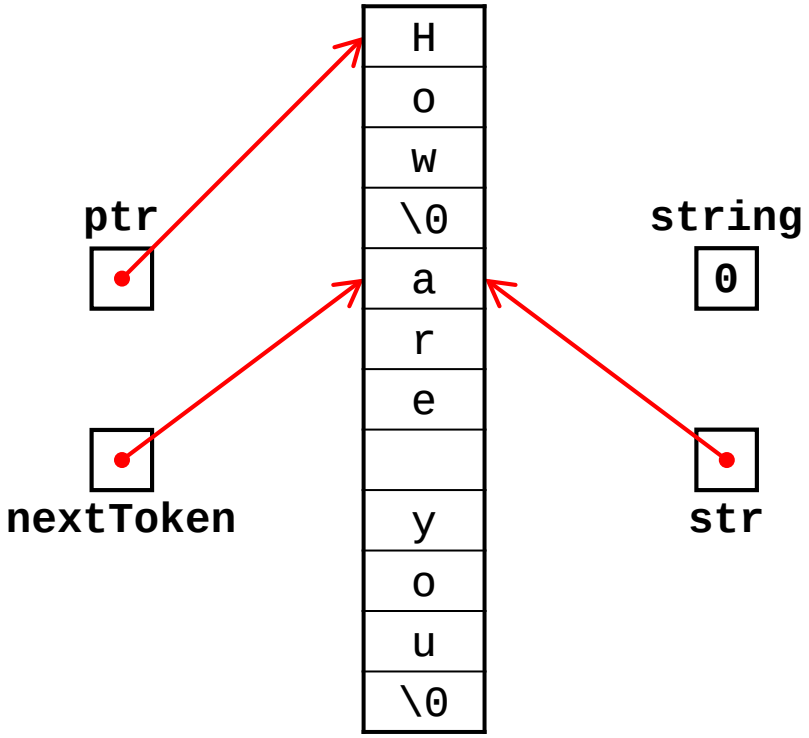
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



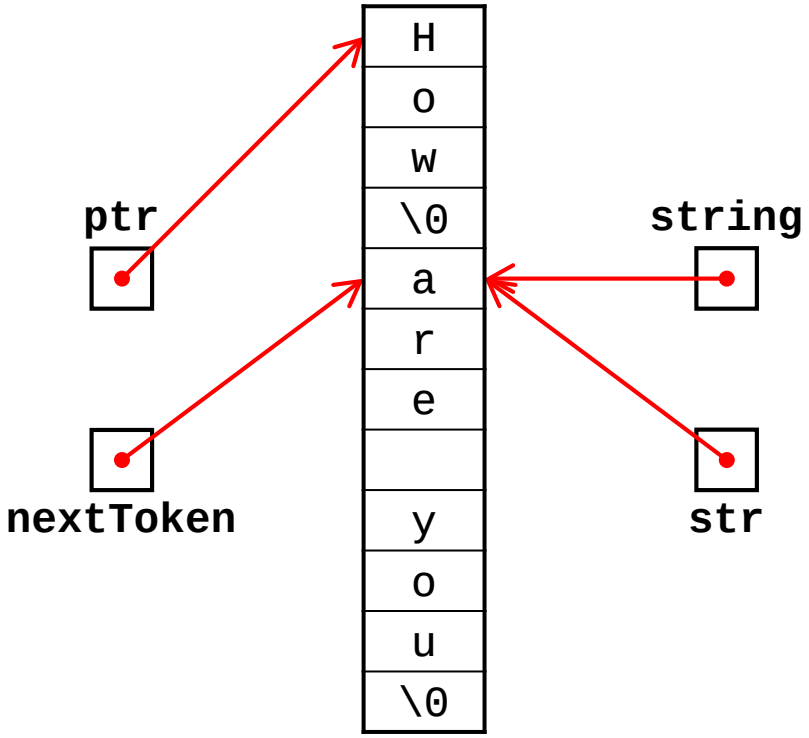
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



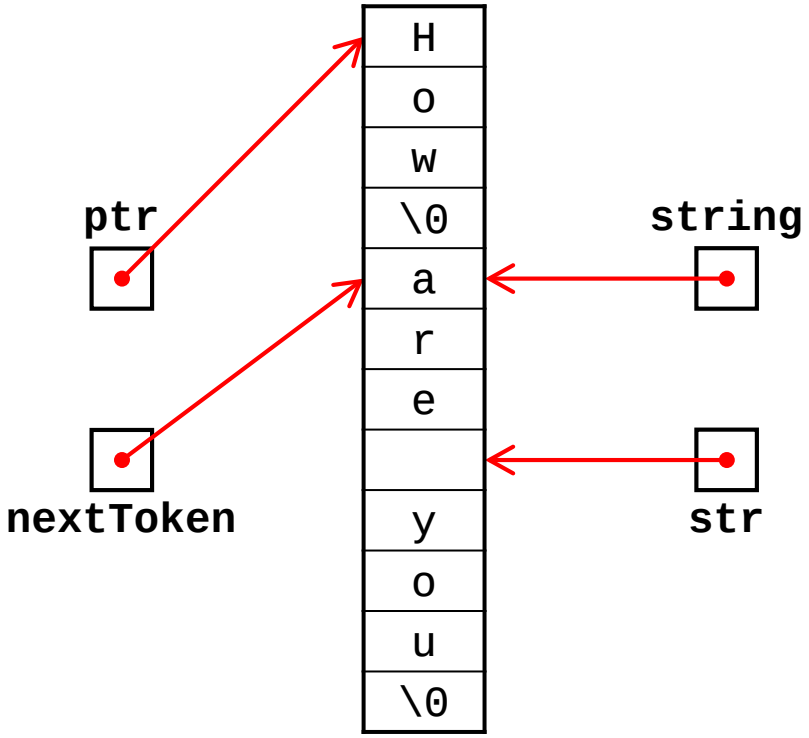
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



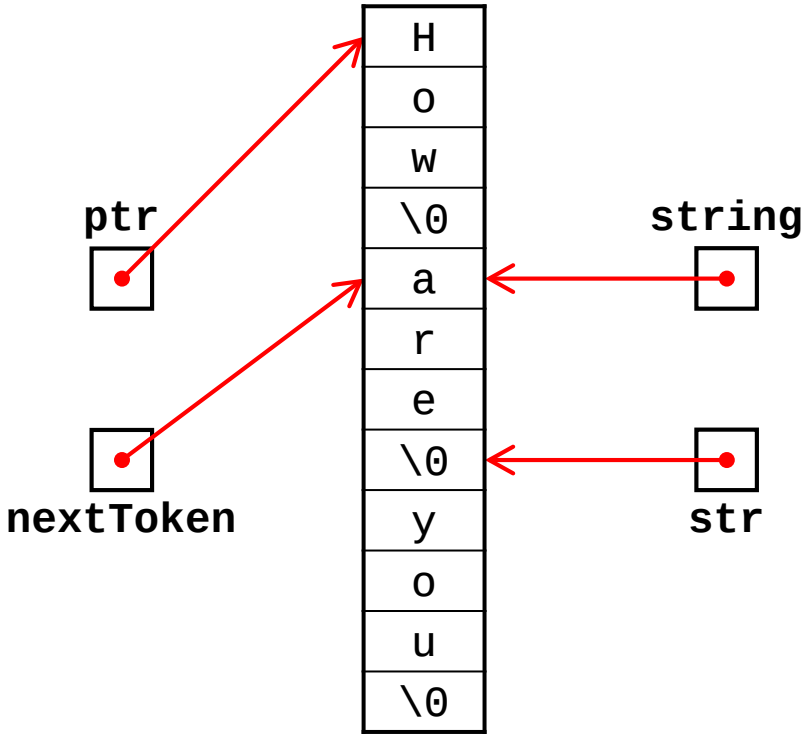
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



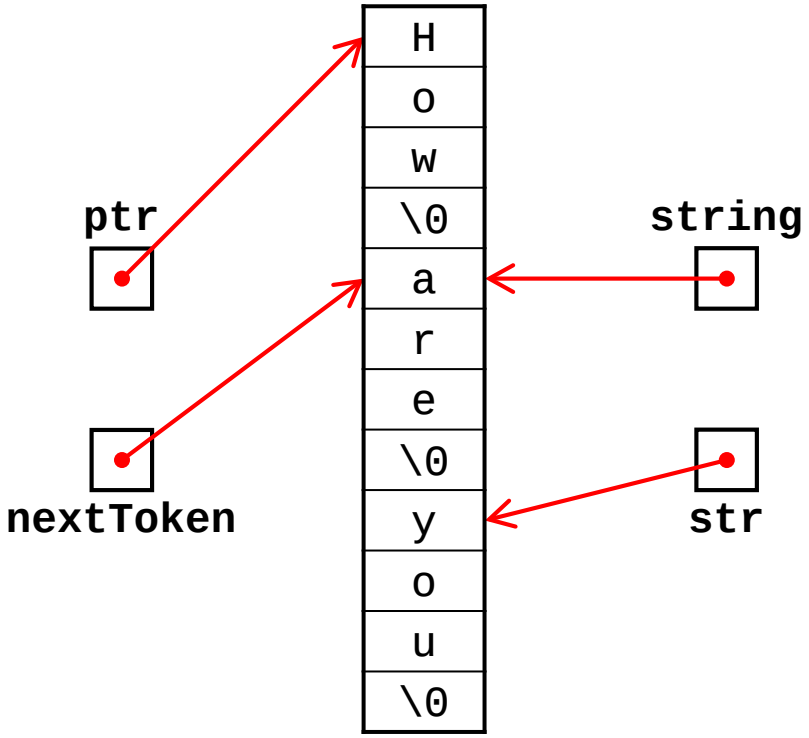
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



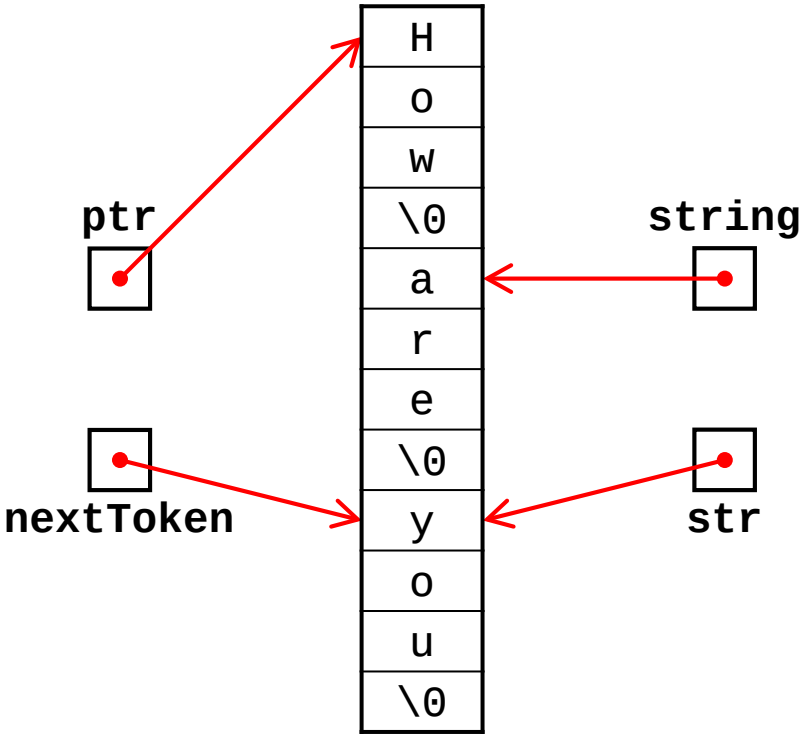
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



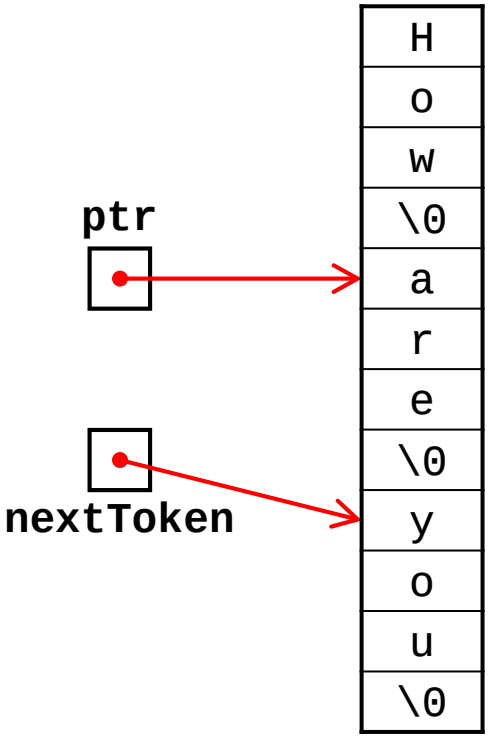
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



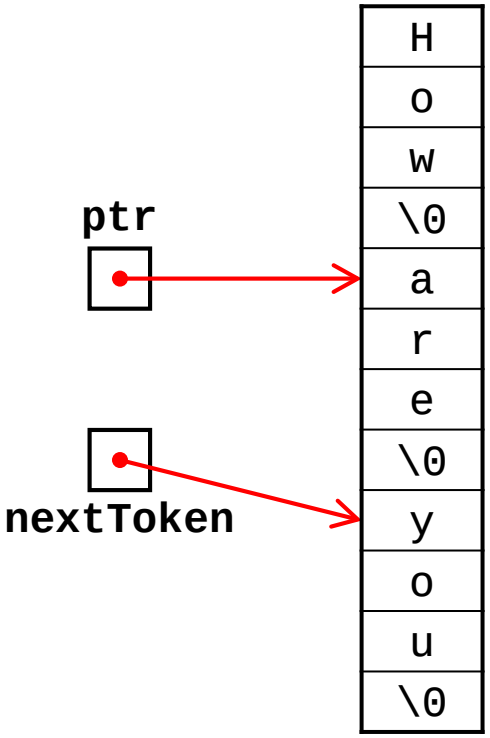
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



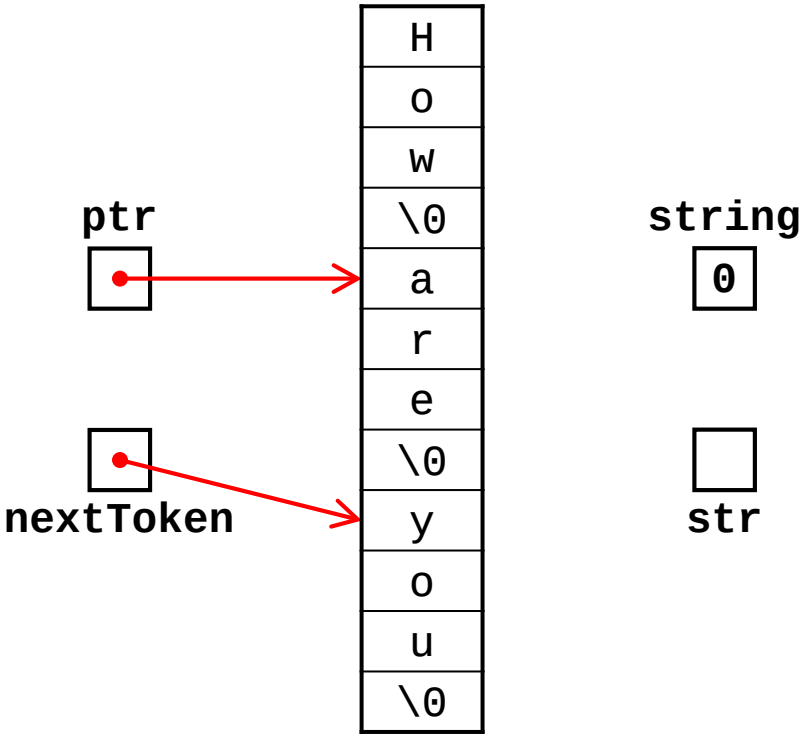
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



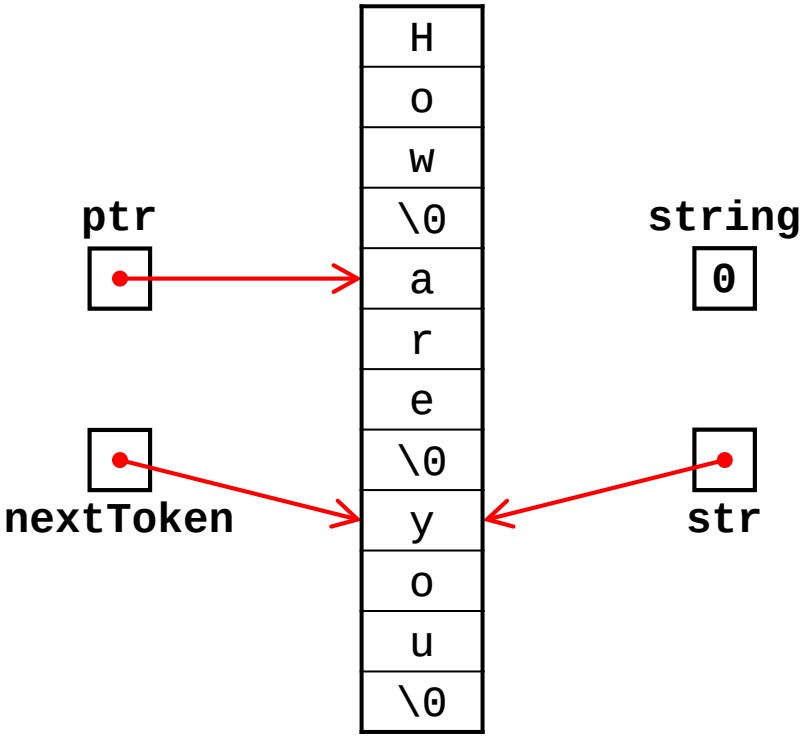
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



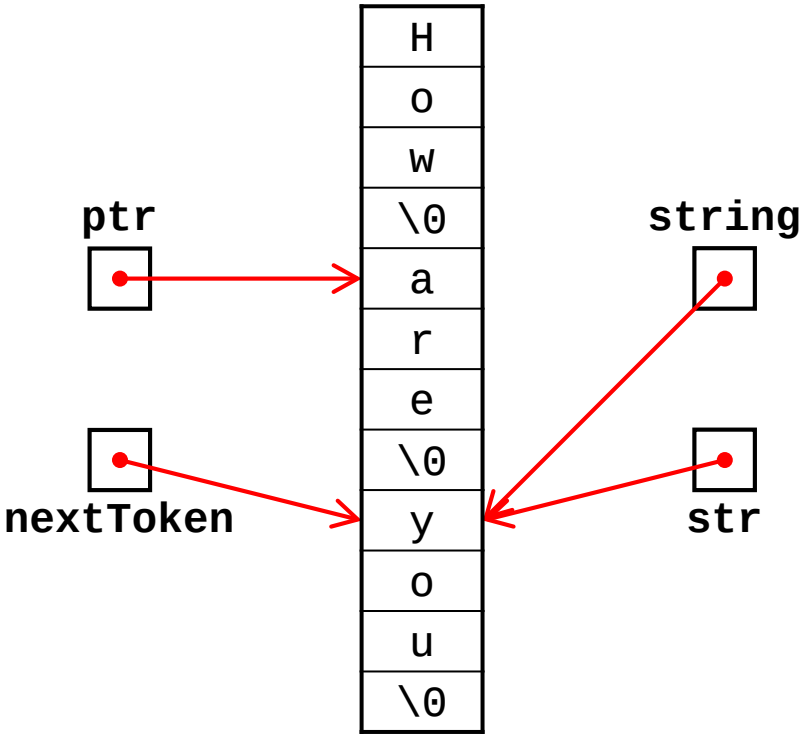
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



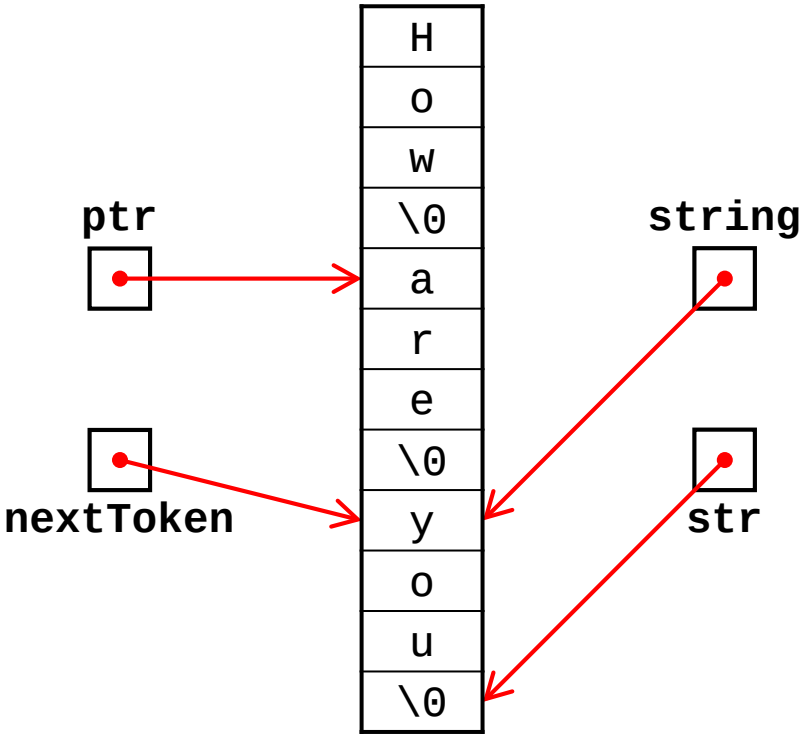
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



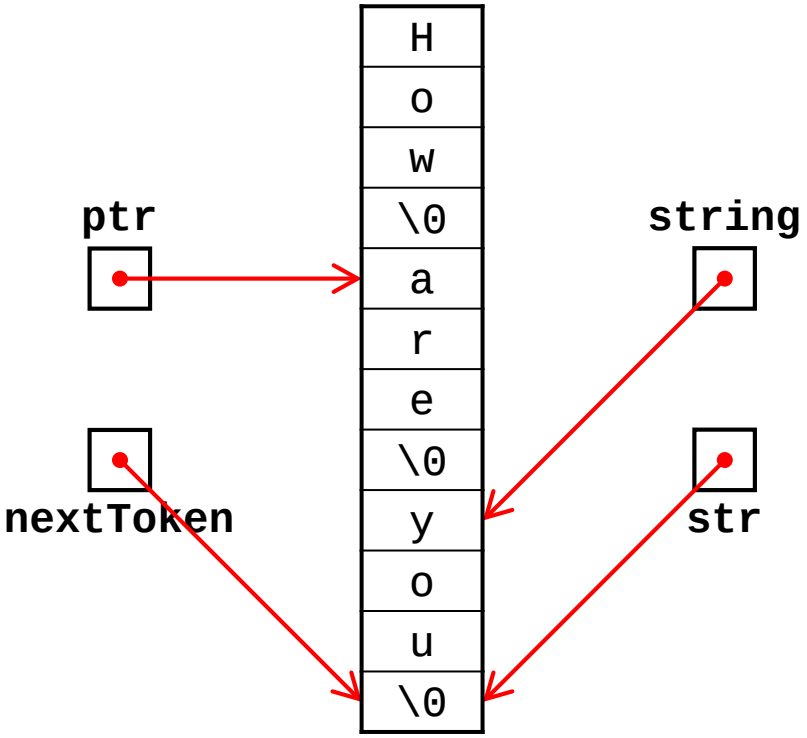
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



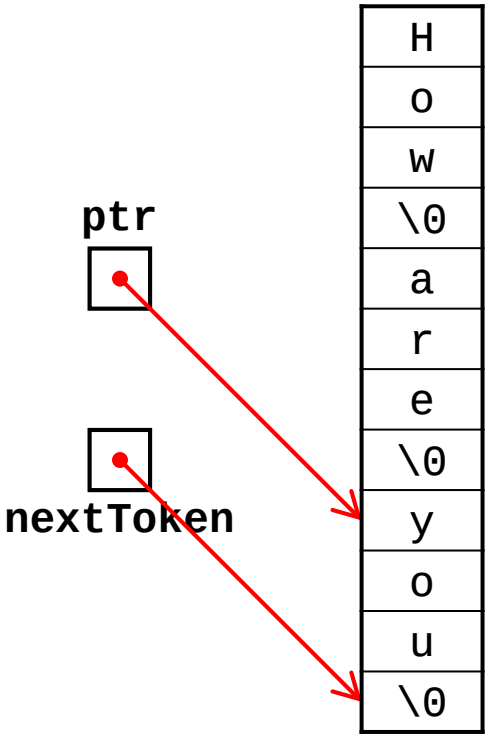
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



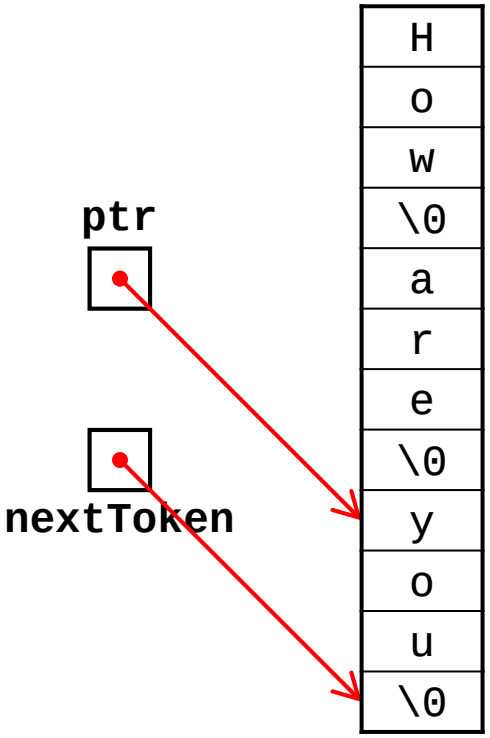
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



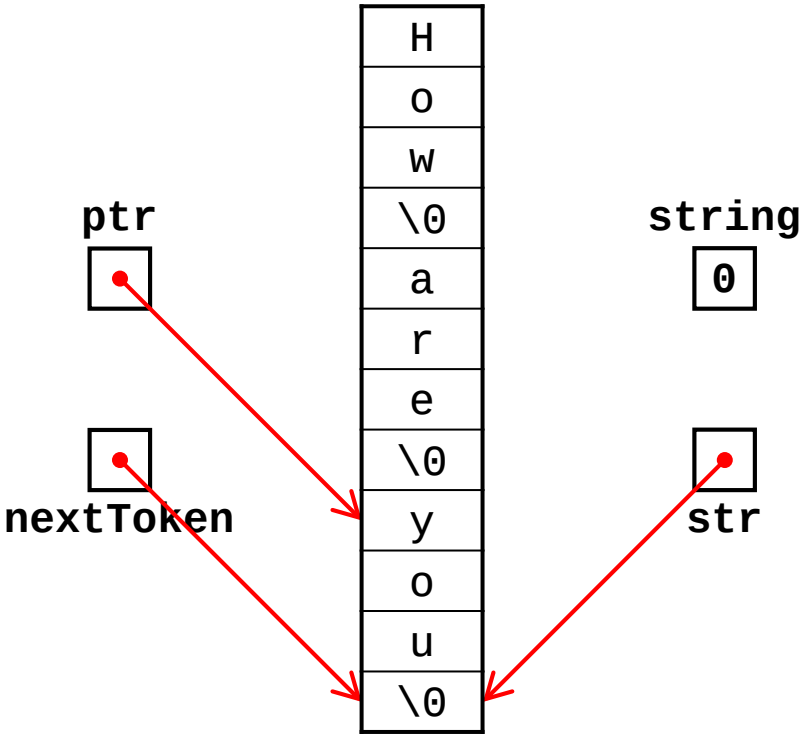
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

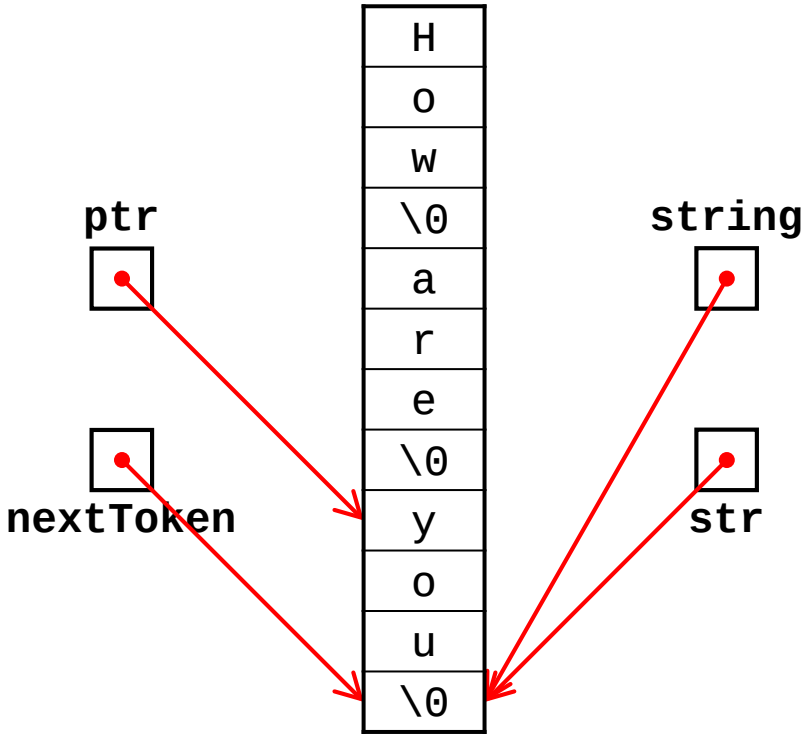
Output



```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

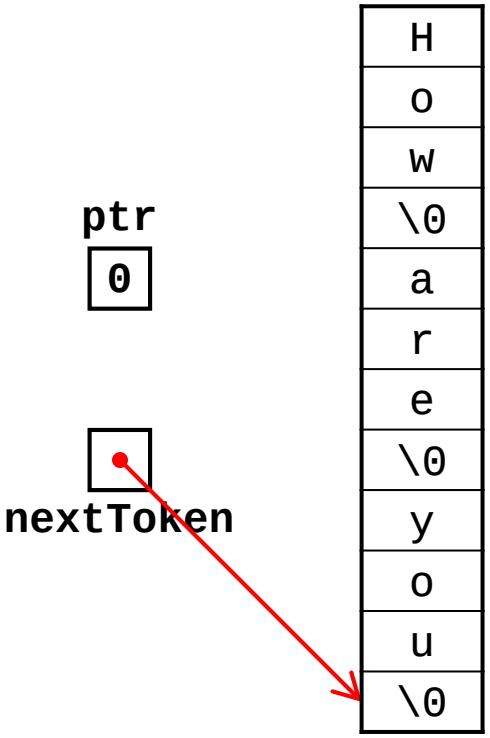
```
How
are
you
```

Output



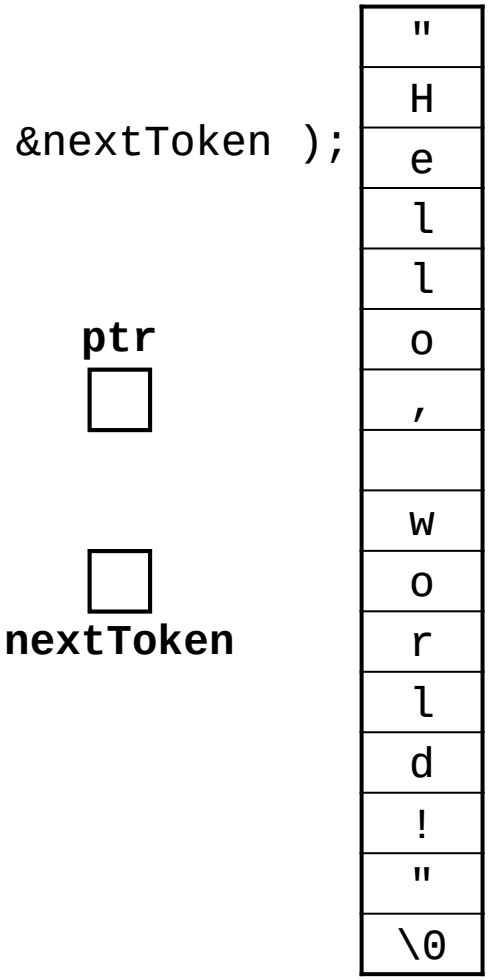
```
int main()
{
   char string[] = "How are you";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, " ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, " ", &nextToken );
   }
}
```

Output



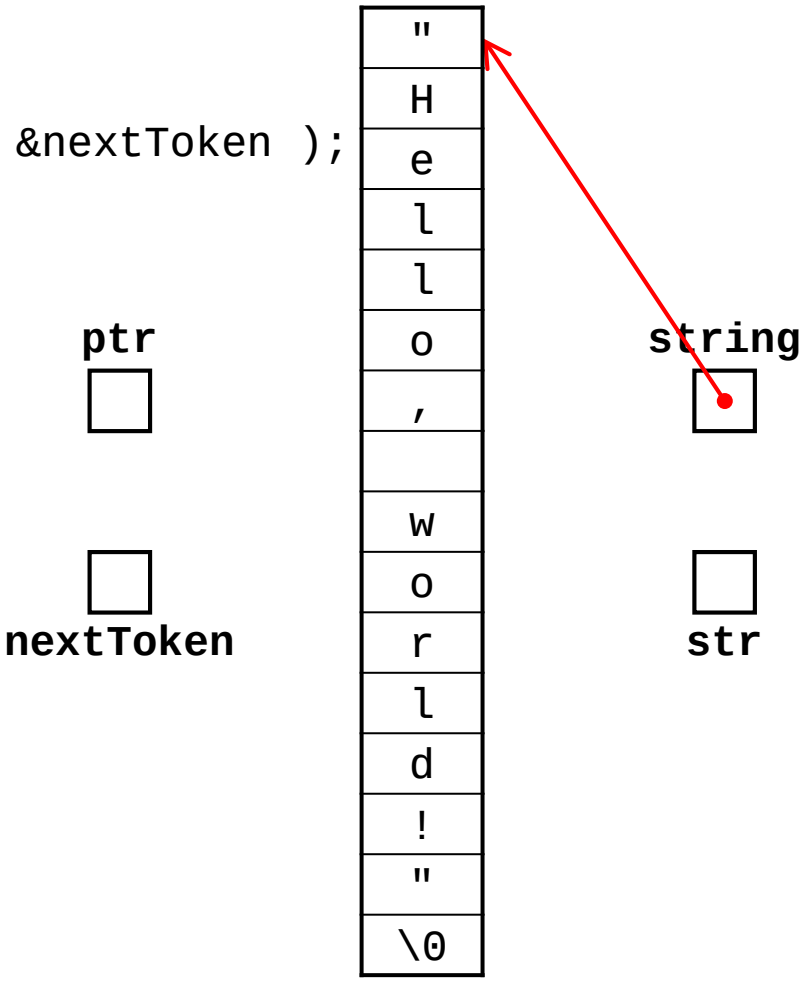
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



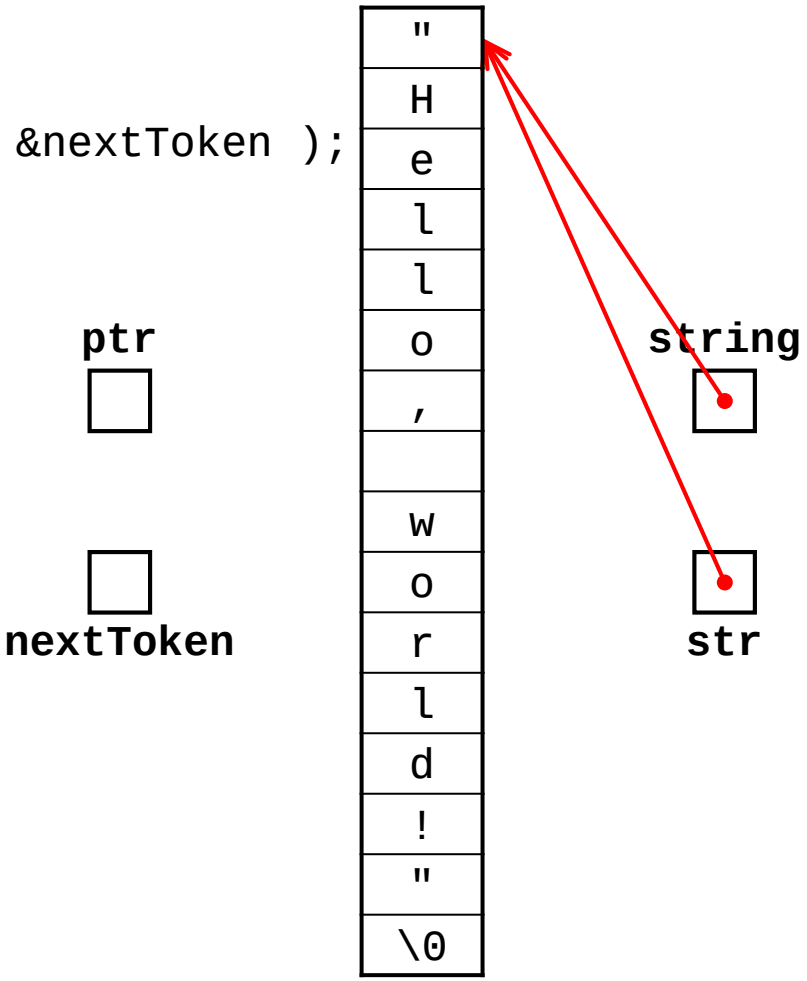
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



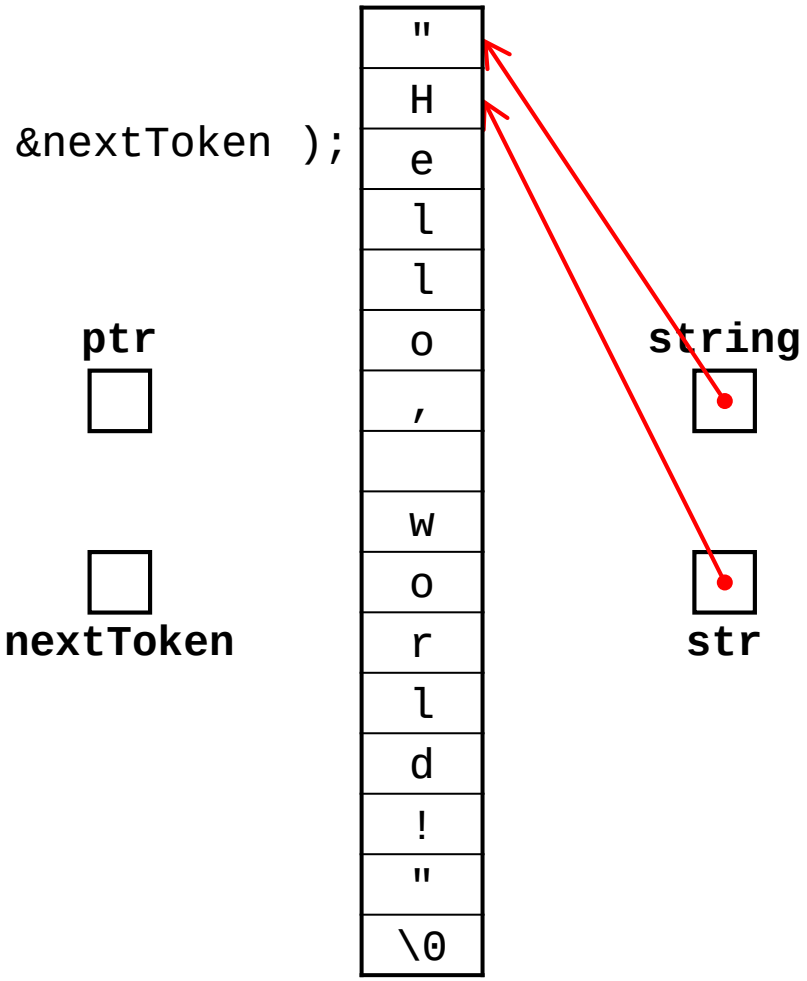
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



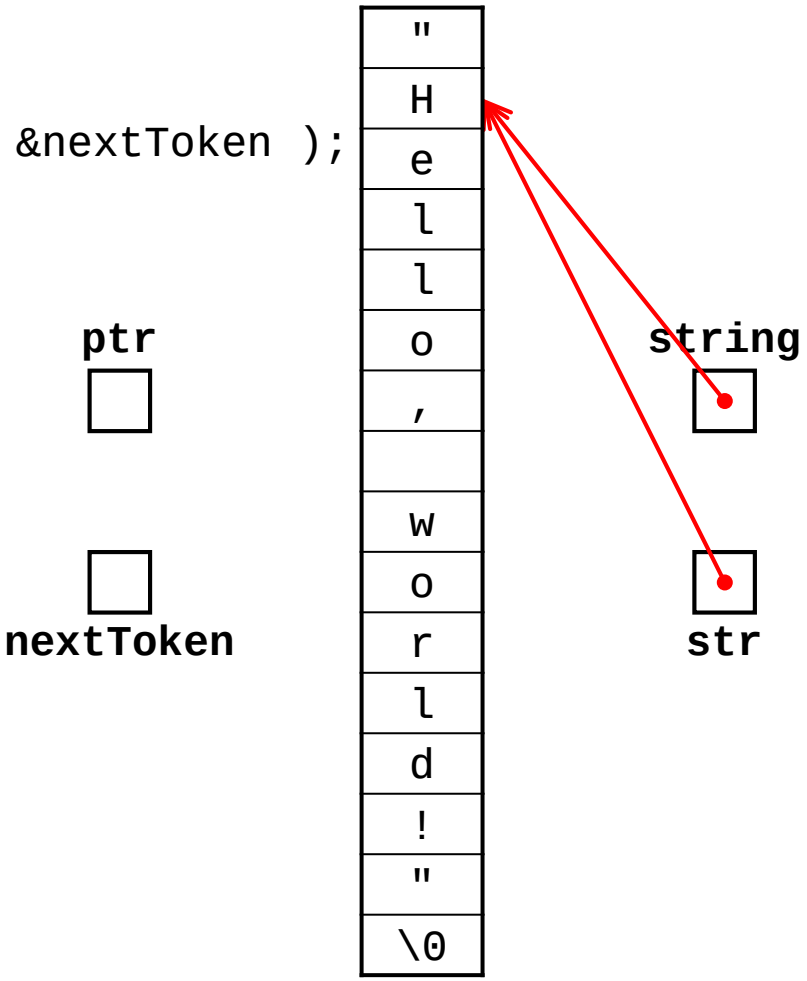
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



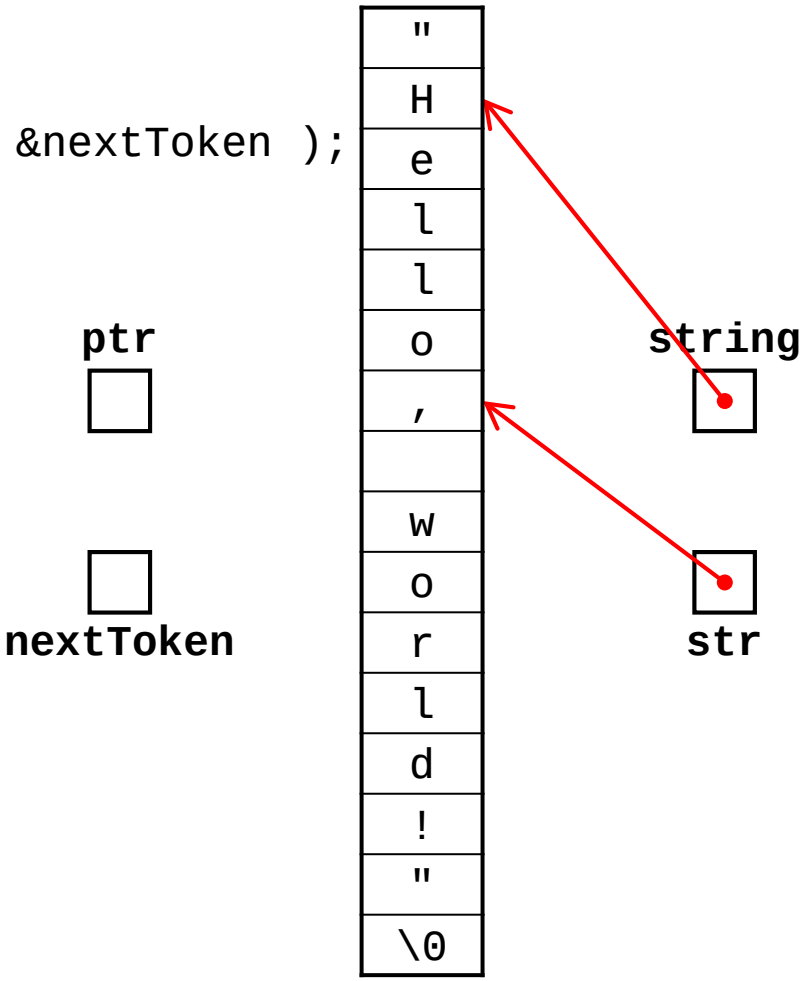
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



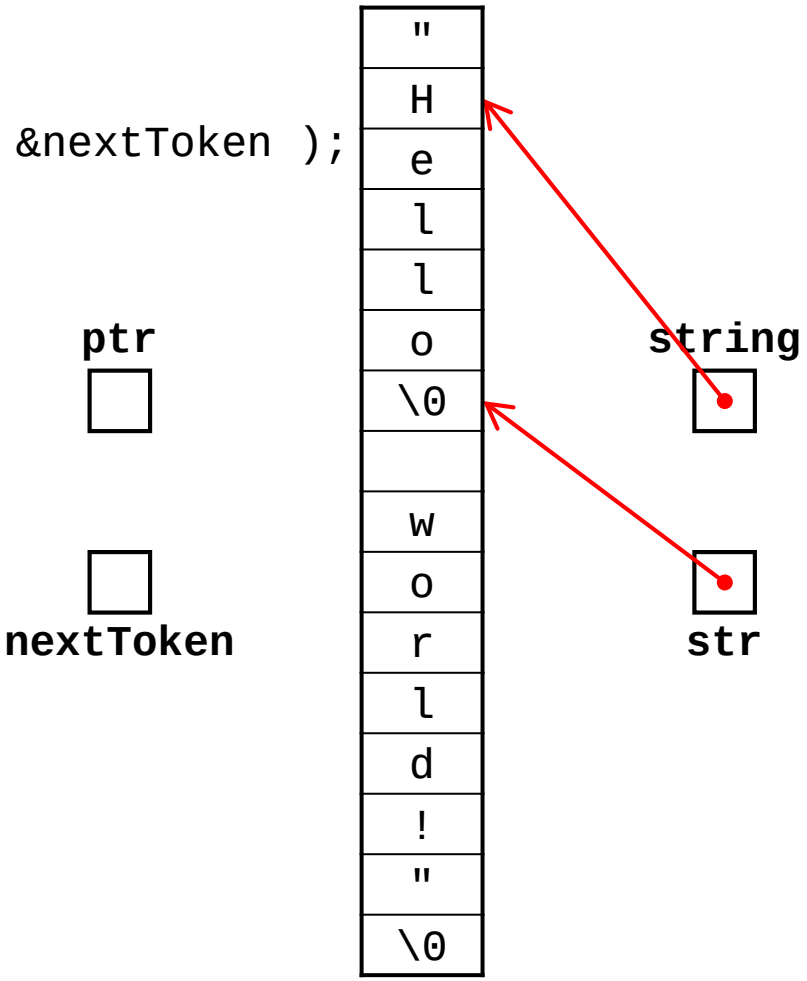
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



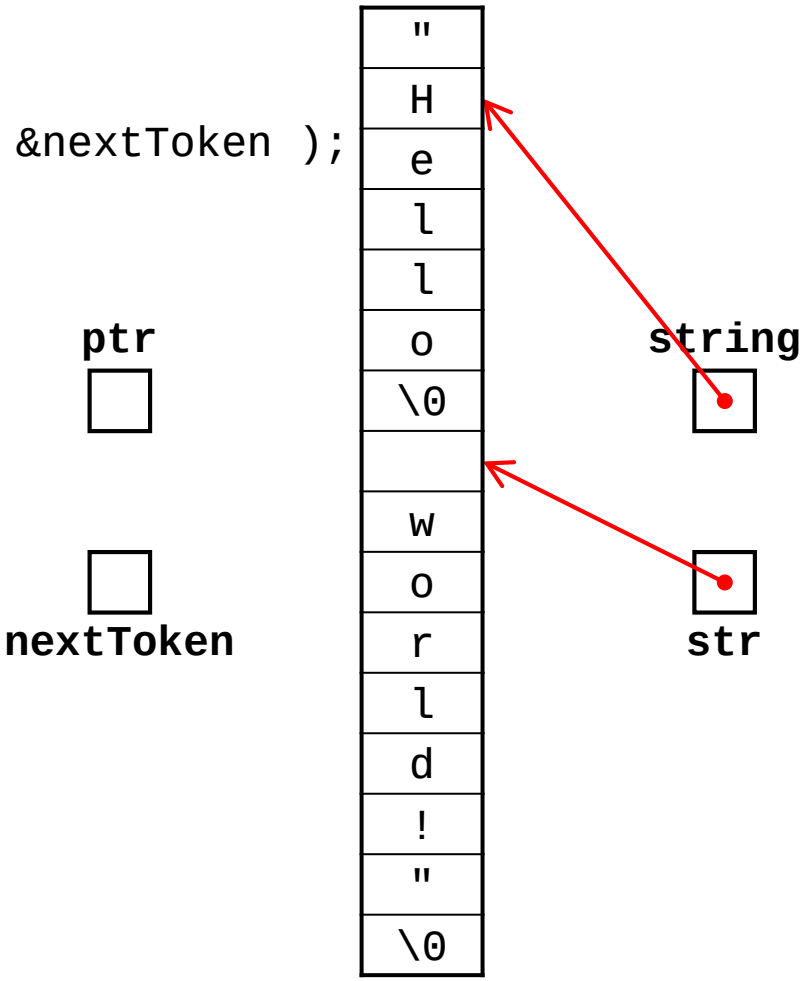
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



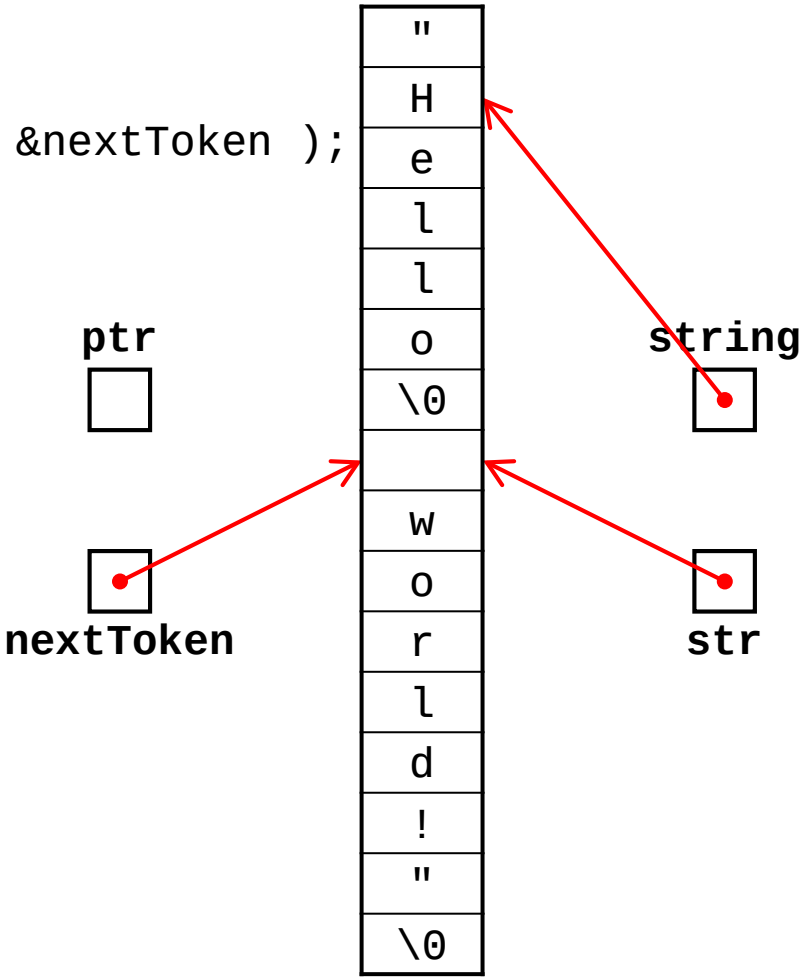
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



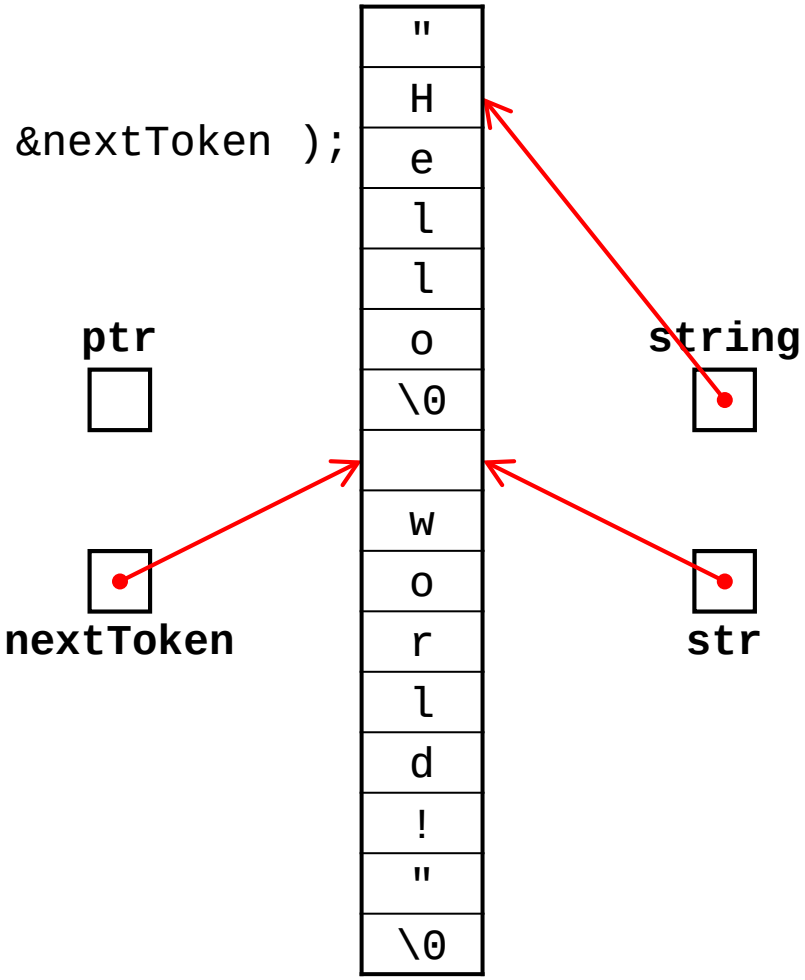
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output



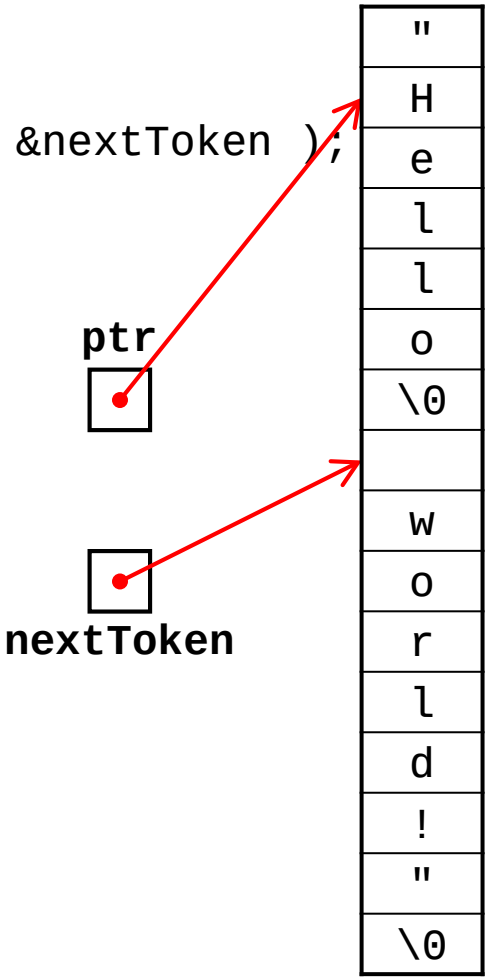
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

Output

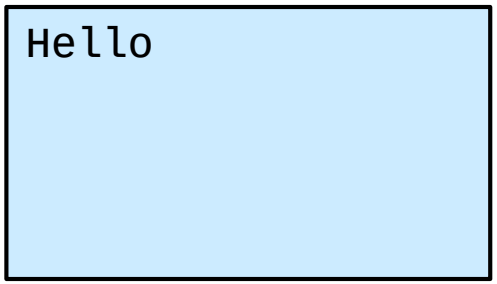


```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

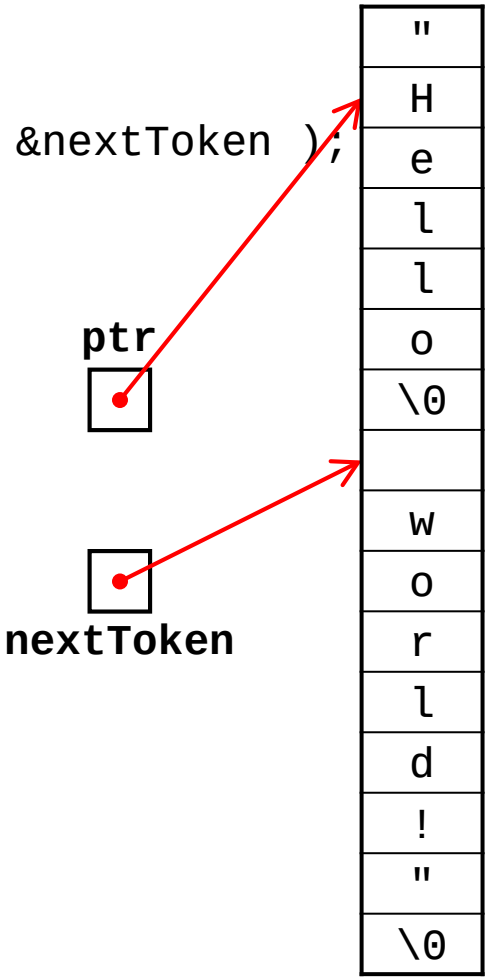
Output



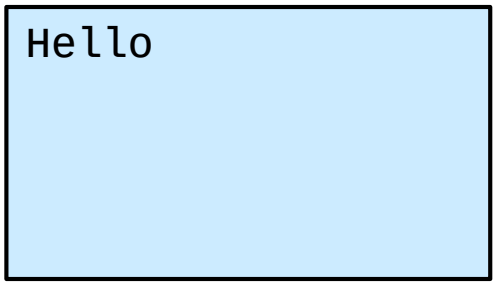
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



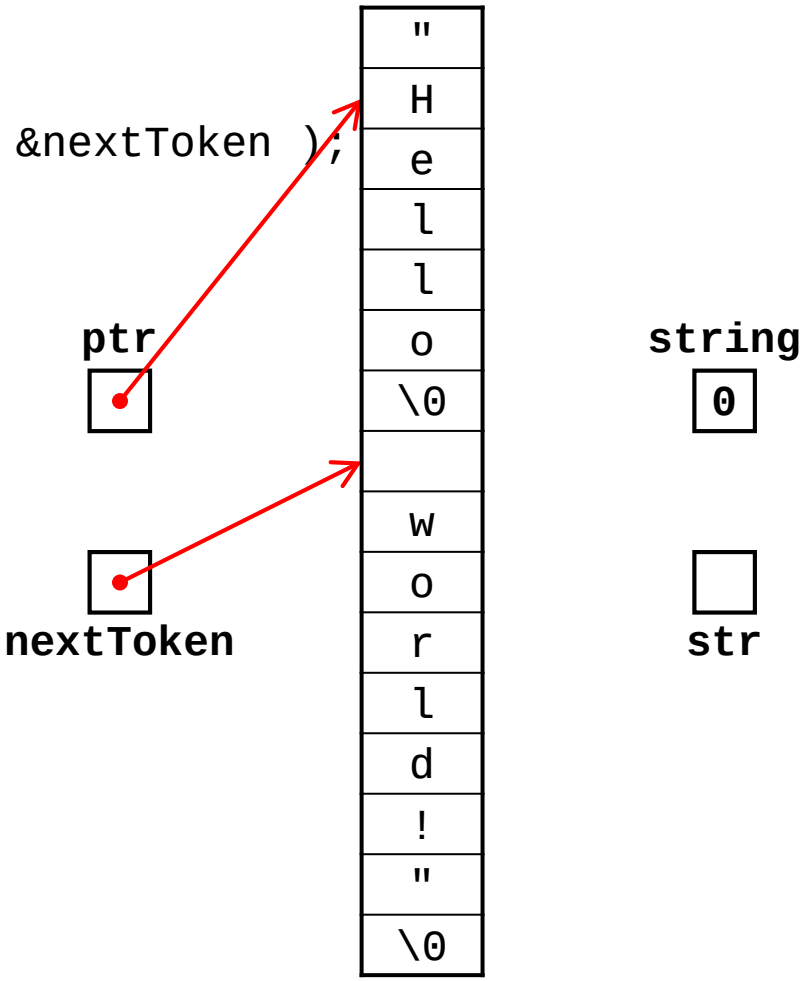
Output



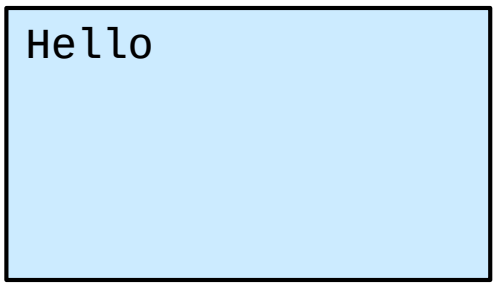
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



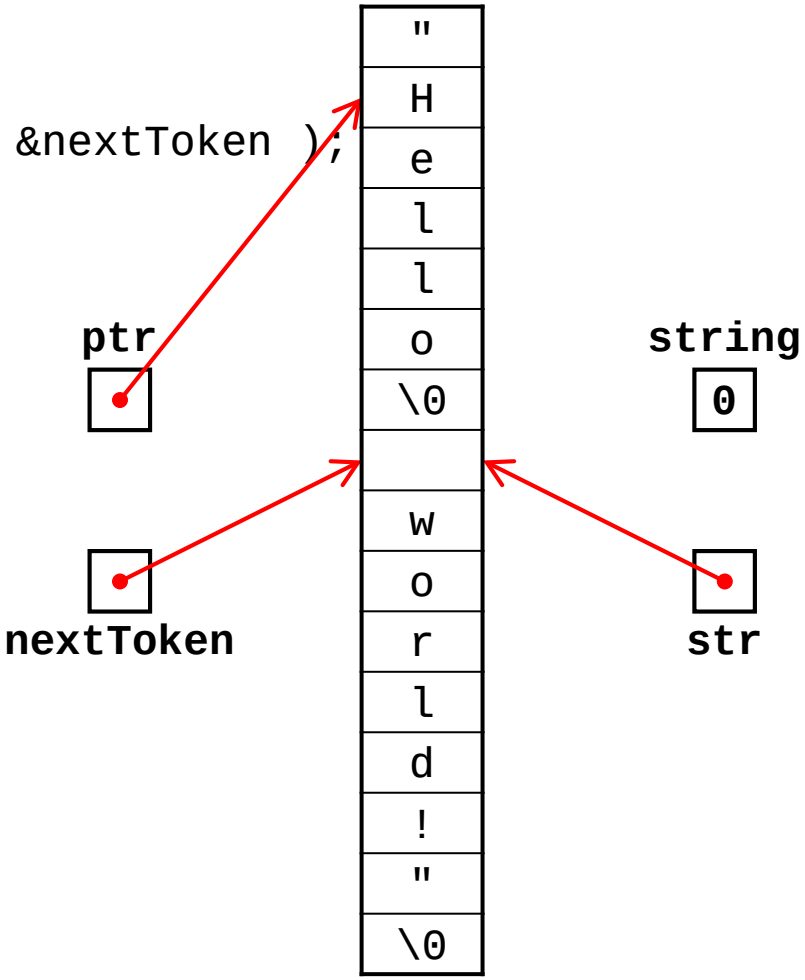
Output



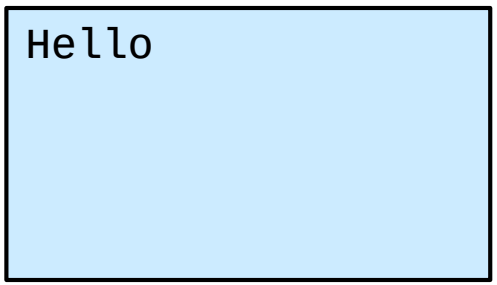
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



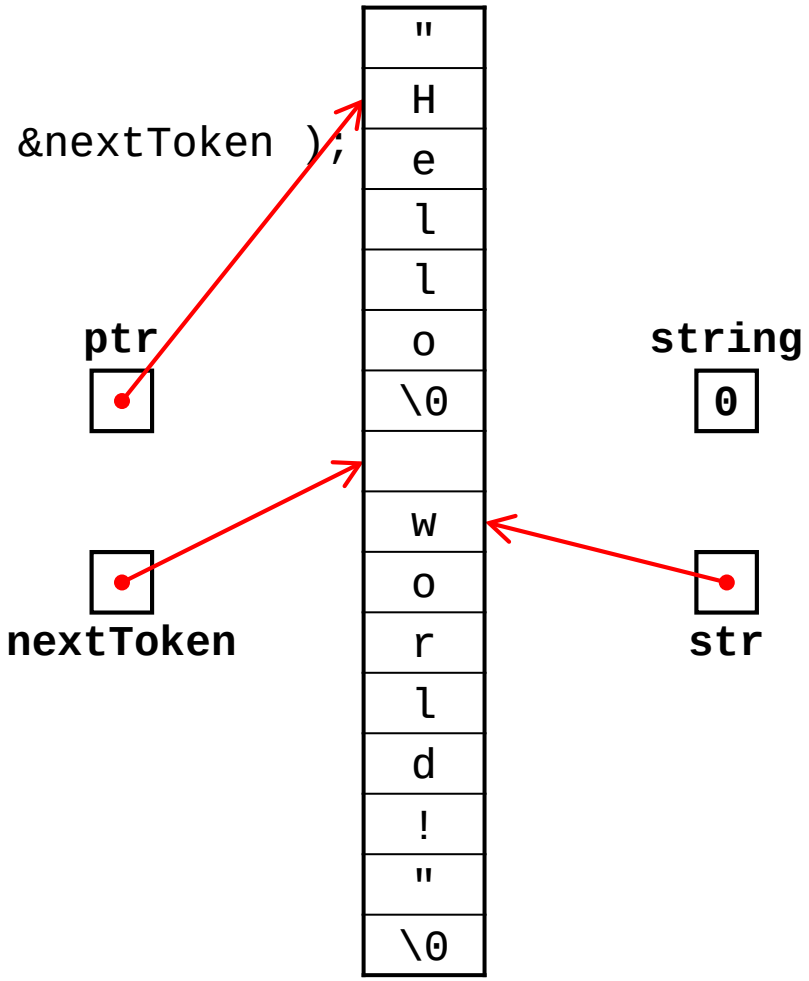
Output



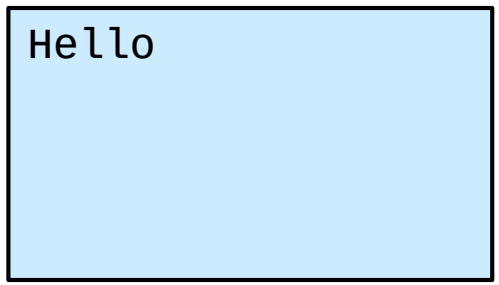
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



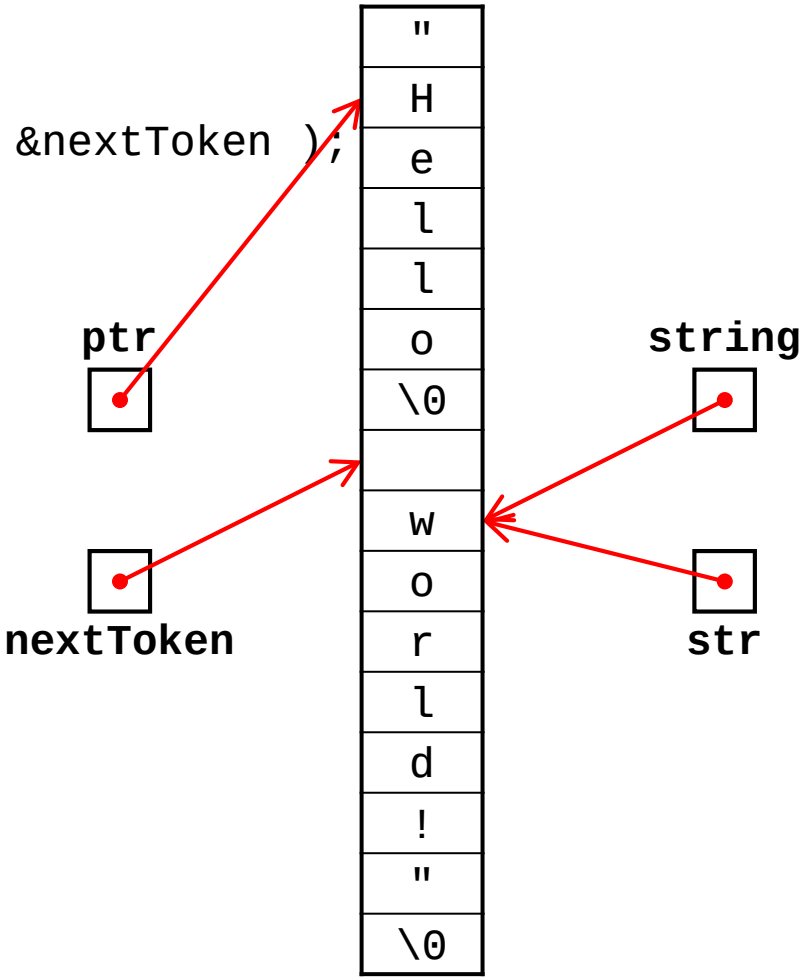
Output



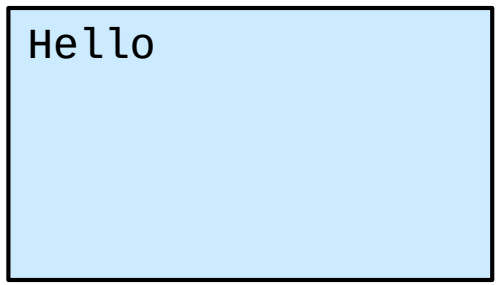
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



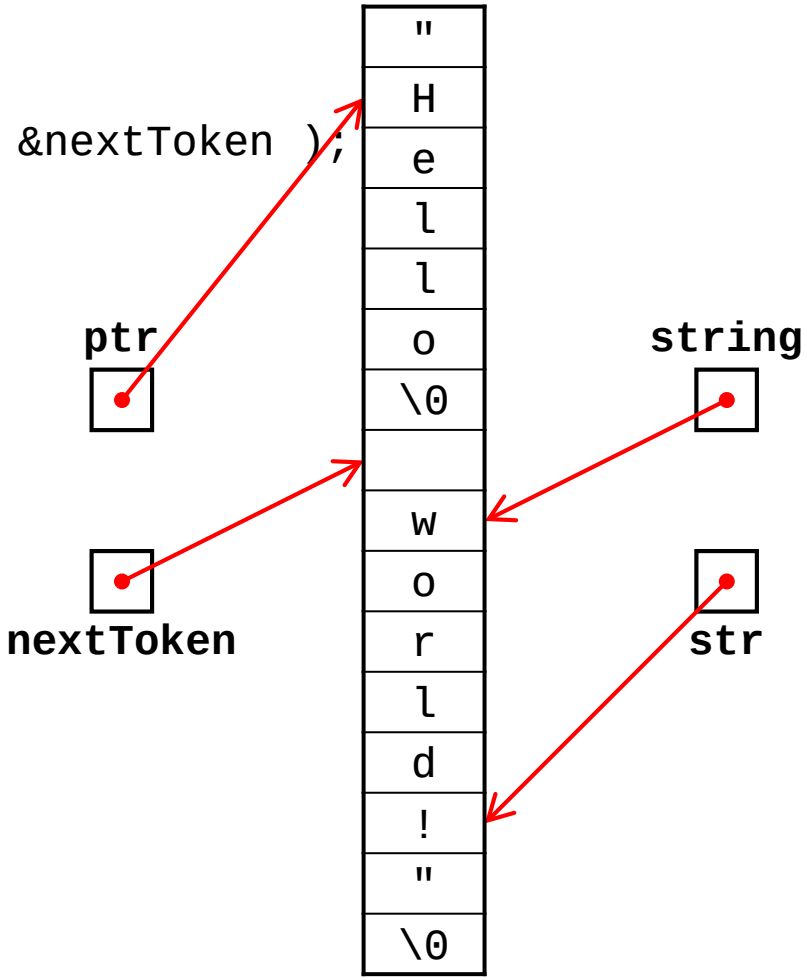
Output



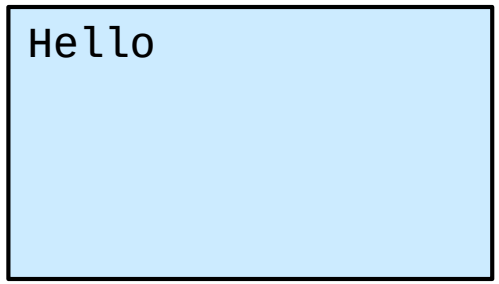
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



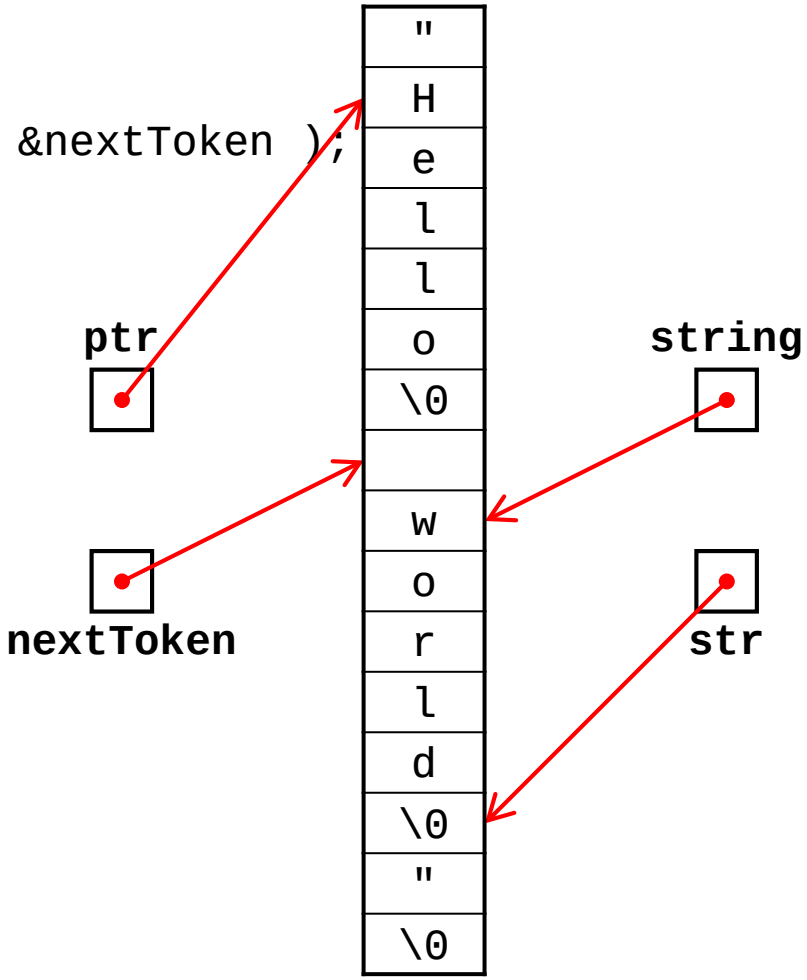
Output



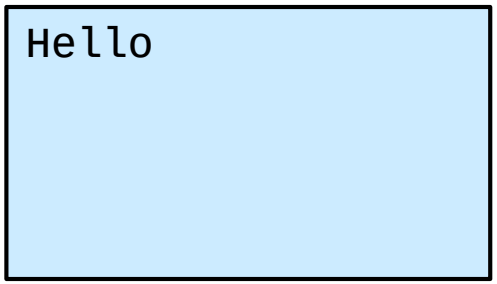
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



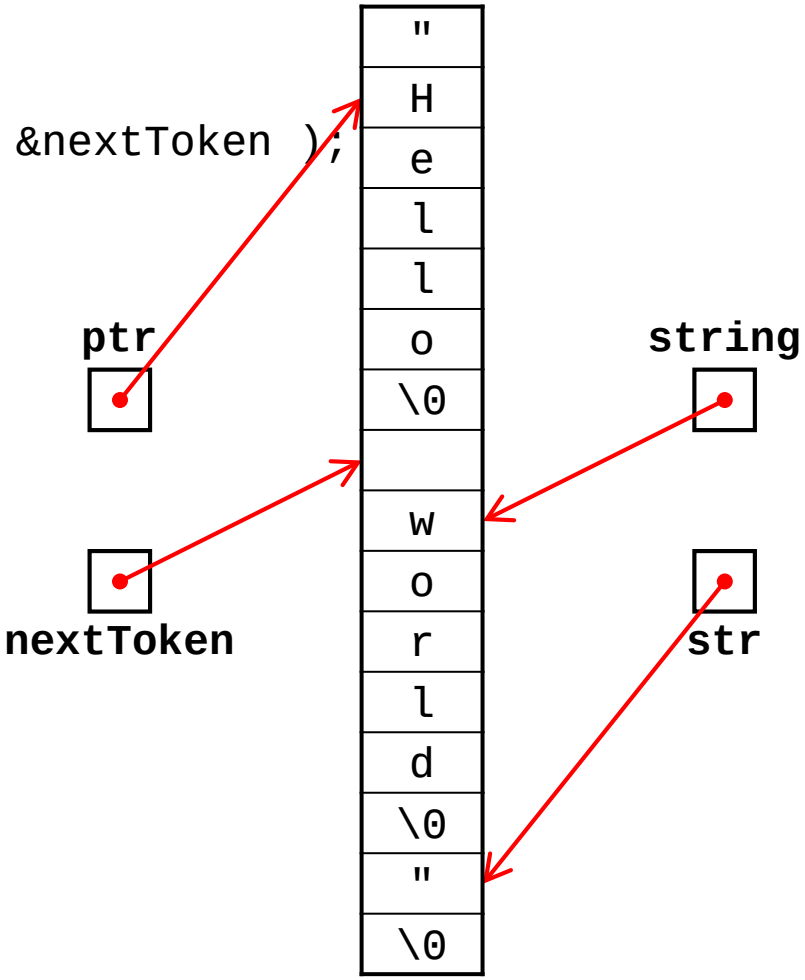
Output



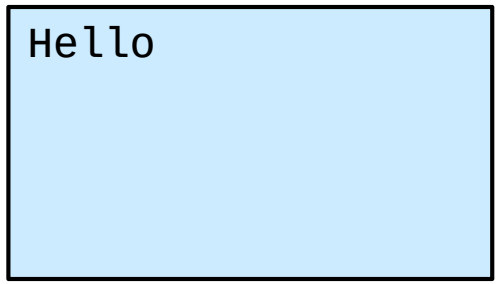
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



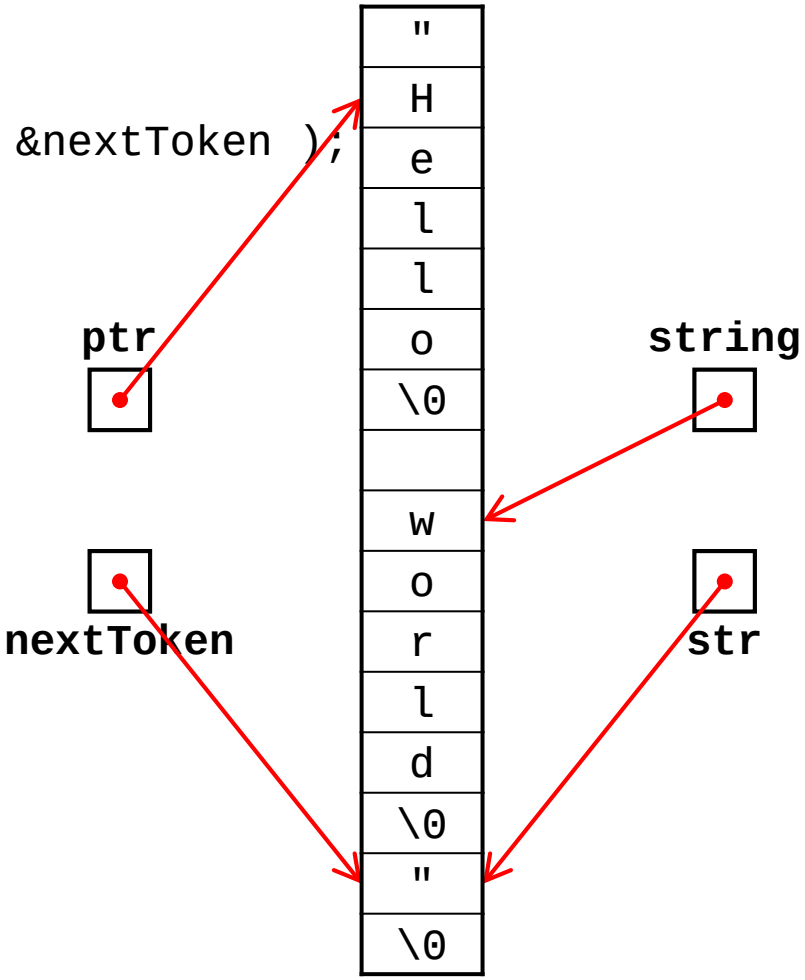
Output



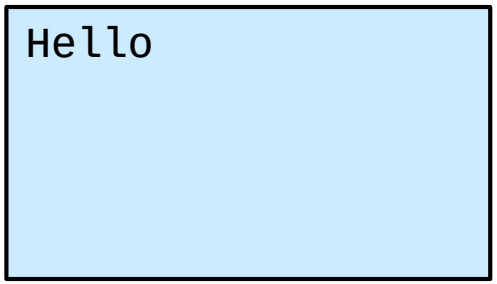
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



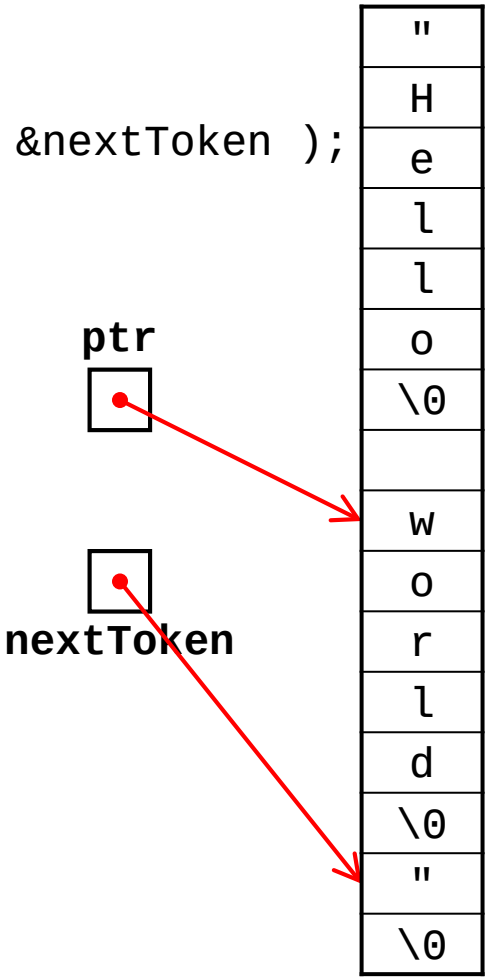
Output



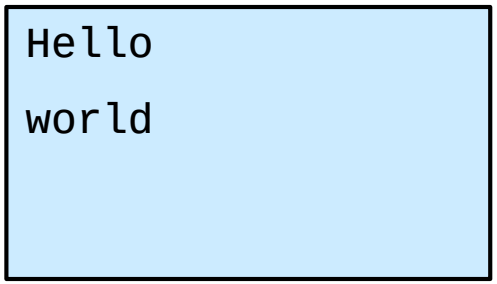
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



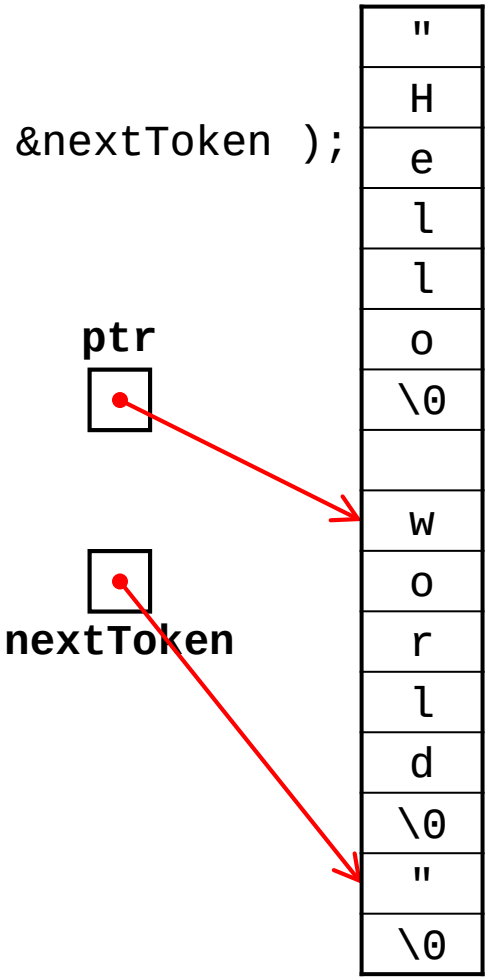
Output



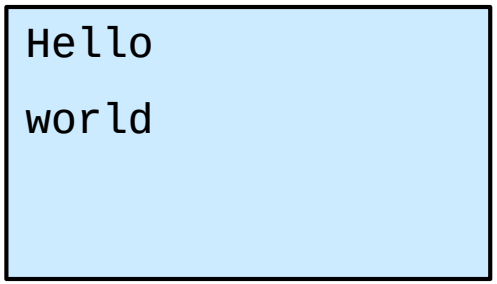
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



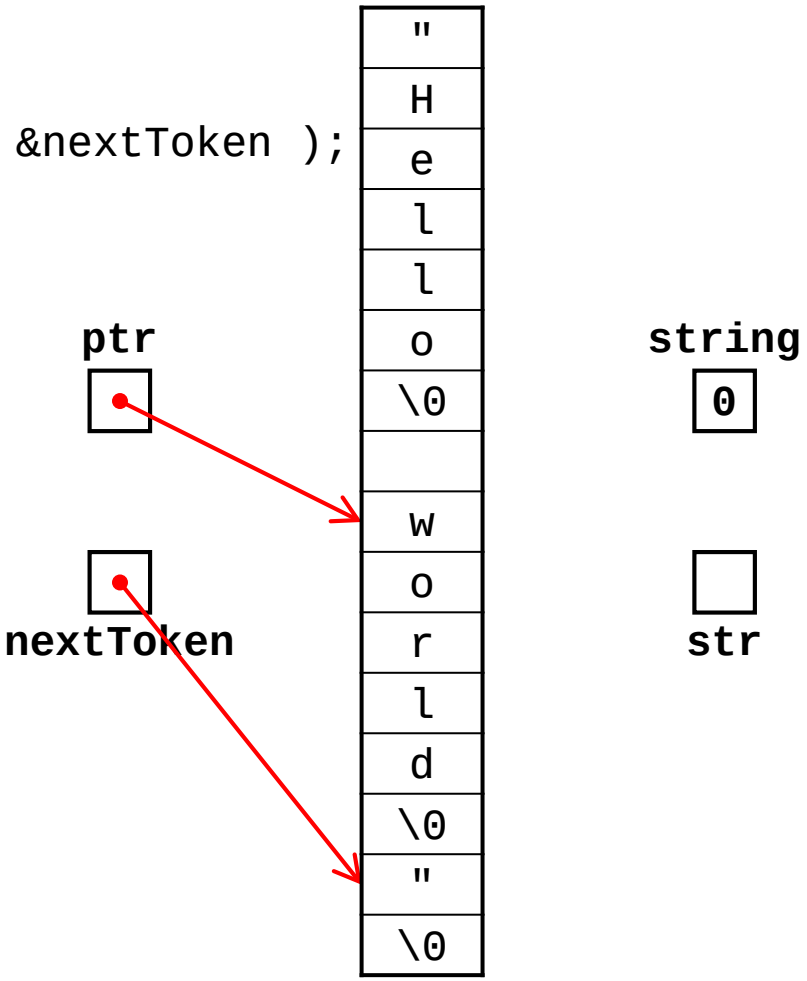
Output



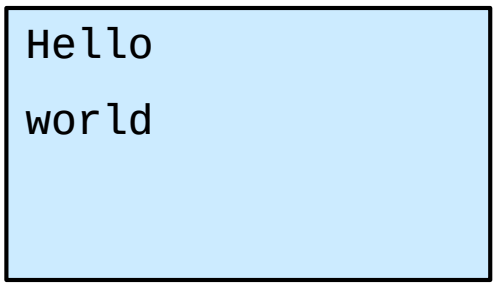
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



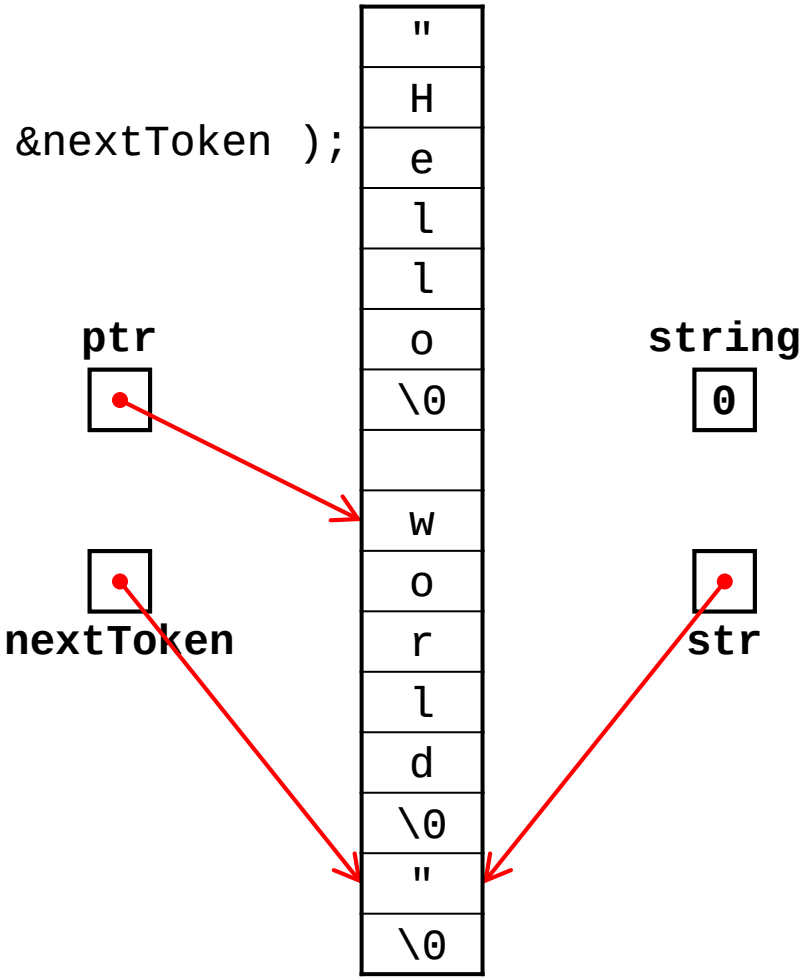
Output



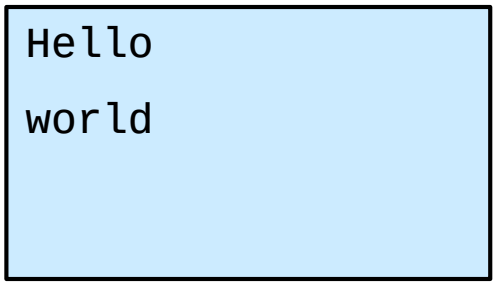
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



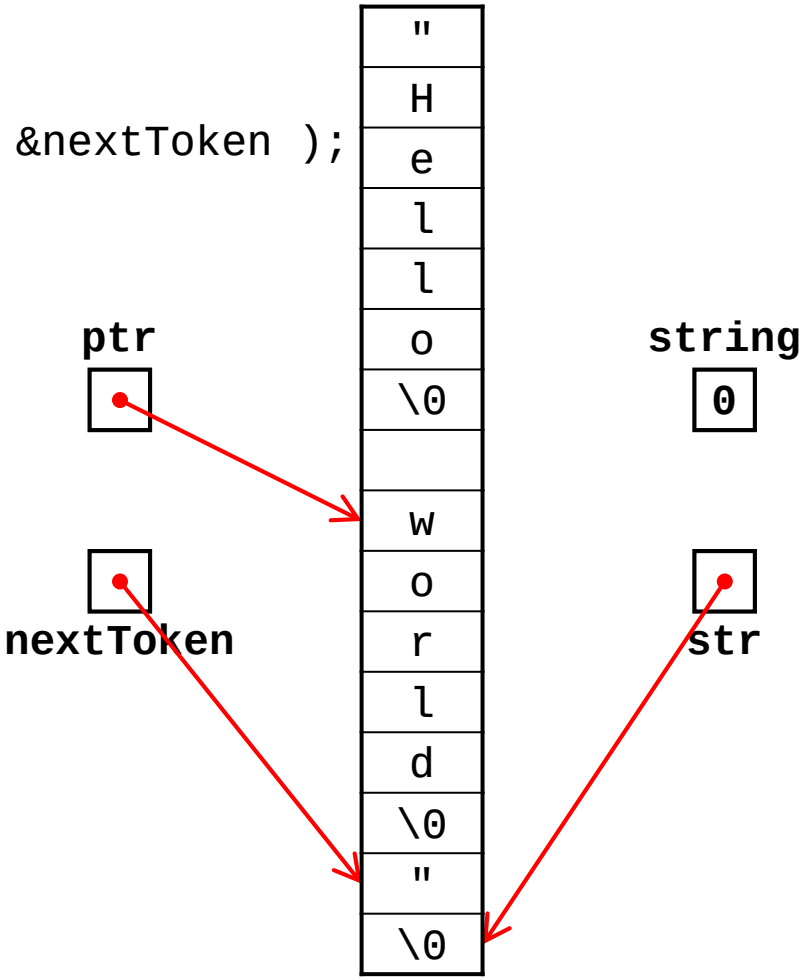
Output



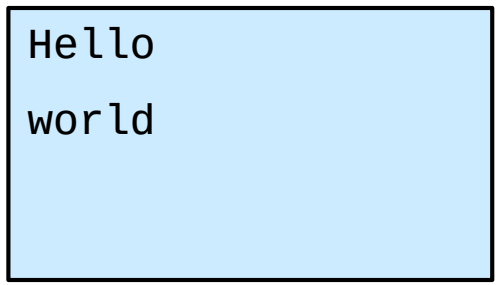
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



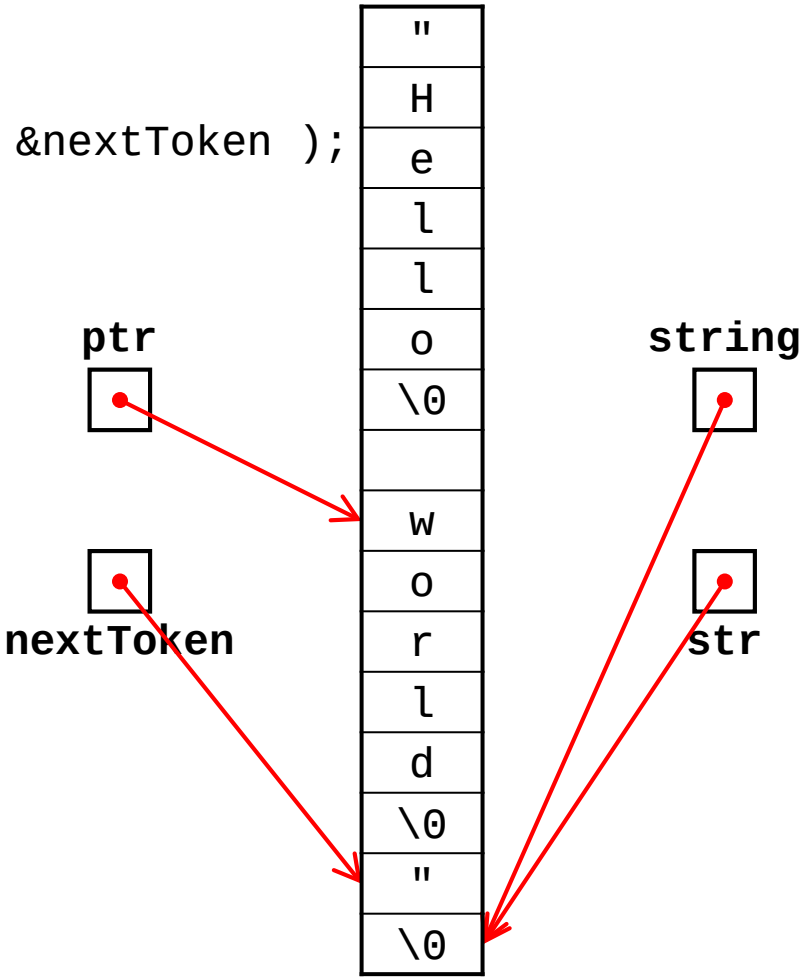
Output



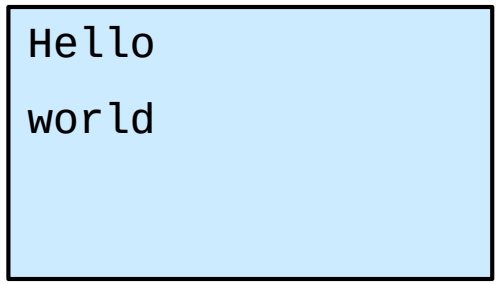
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



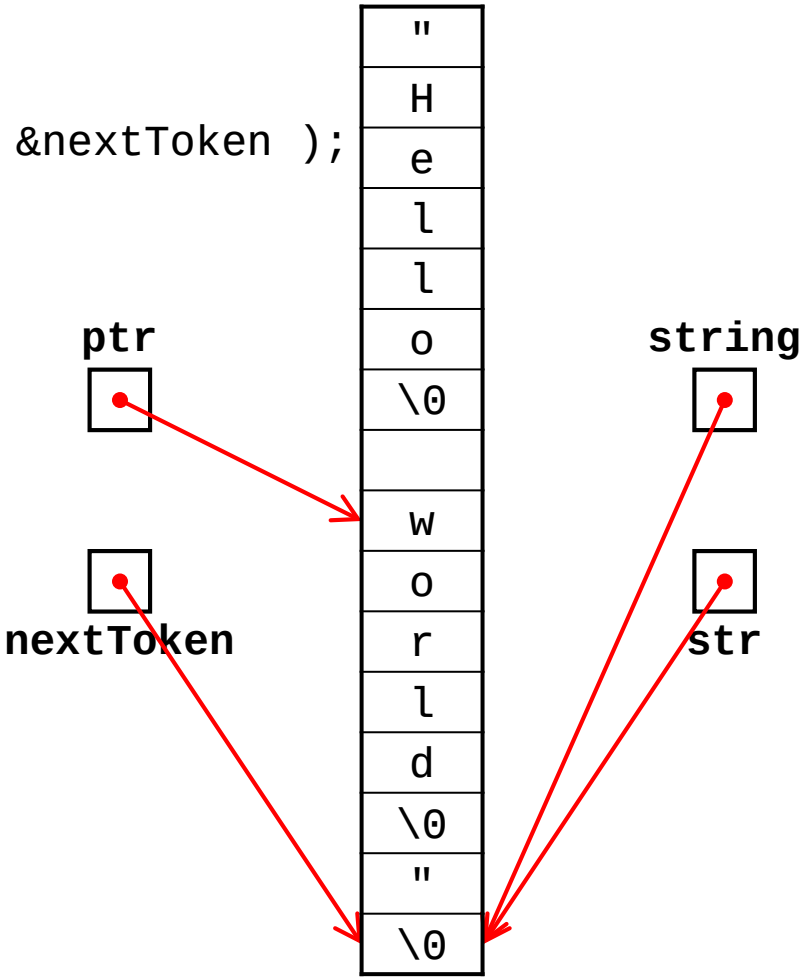
Output



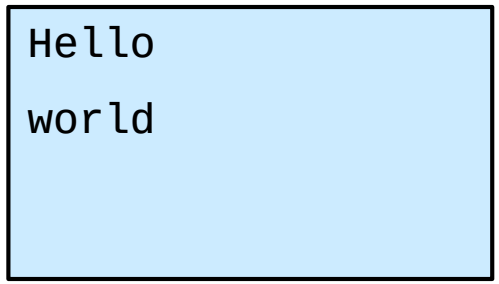
```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```



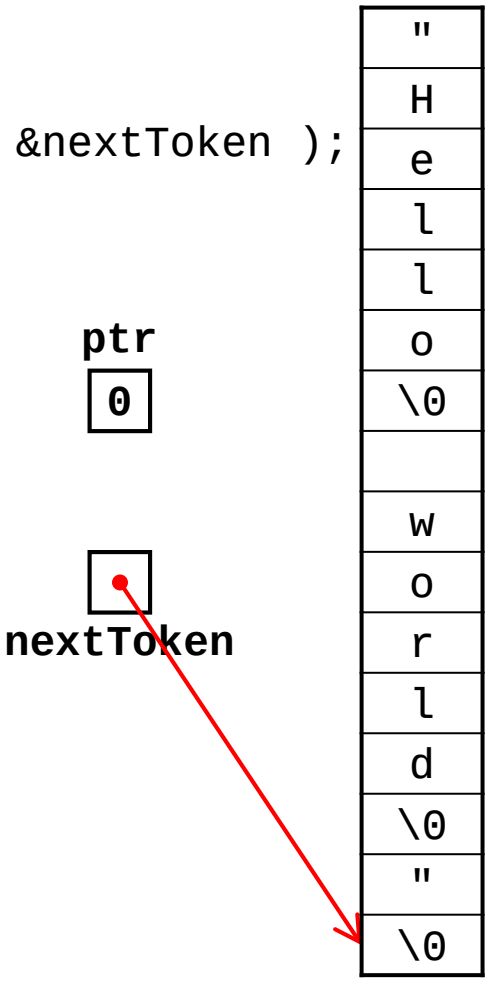
Output



```
int main()
{
   char string[] = "\"Hello, world!\"";
   char *nextToken;
   char *ptr;
   ptr = strtok_s( string, "\",! ", &nextToken );
   while( ptr != nullptr )
   {
      cout << ptr << '\n';
      ptr = strtok_s( nullptr, "\",! ", &nextToken );
   }
}
```

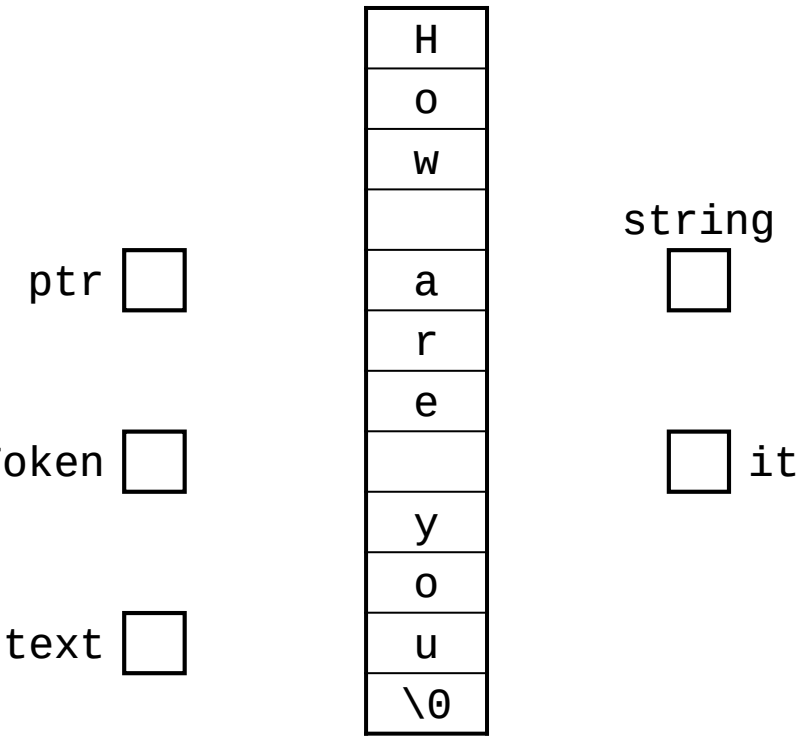


Output



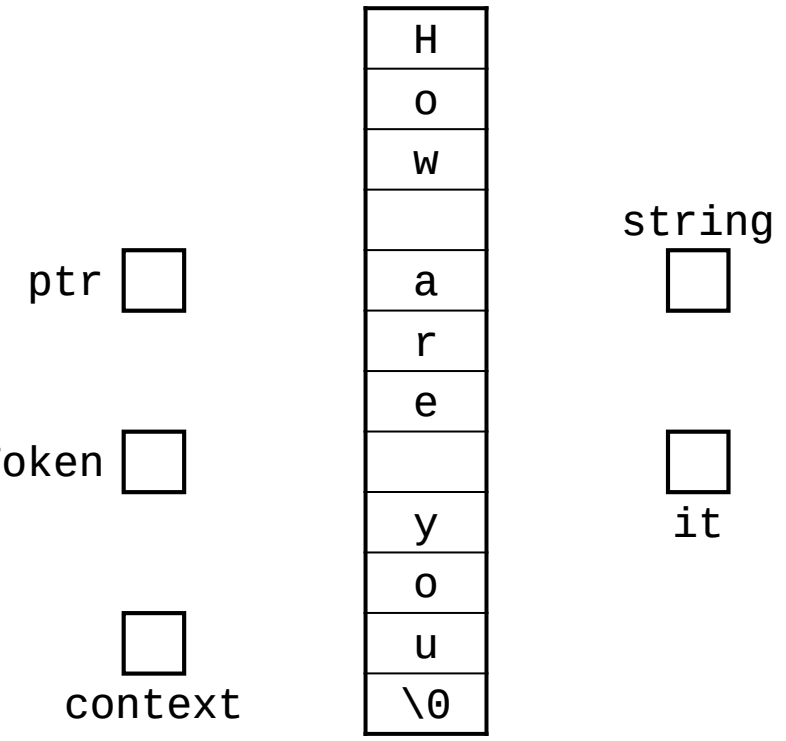
```
char *strtok_s( char *string, const char *control, char **context )
{
   char *it = ( string == nullptr ) ? *context : string;

   // Find the next nondelimiter
   while( belong( *it, control ) )
      it++;

   char * const tokenFirst = it;

   // Find the next delimiter
   for( ; *it != '\0'; it++ )
      if( belong( *it, control ) )
      {
         *( it++ ) = '\0';
         break;
      }

   *context = it;

   return ( tokenFirst == it ) ? nullptr : tokenFirst;
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
   char *it = ( string == nullptr ) ? *context : string;
   // Find the next nondelimiter
   while( belong( *it, control ) )
      it++;
   char * const tokenFirst = it;
   // Find the next delimiter
   for( ; *it != '\0'; it++ )
      if( belong( *it, control ) )
      {
         *( it++ ) = '\0';
         break;
      }


   *context = it;


   return ( tokenFirst == it ) ? nullptr : tokenFirst;
}
```




```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != nullptr )
{
   cout << ptr << '\n';
   ptr = strtok_s( nullptr, " ", &nextToken );
}
```

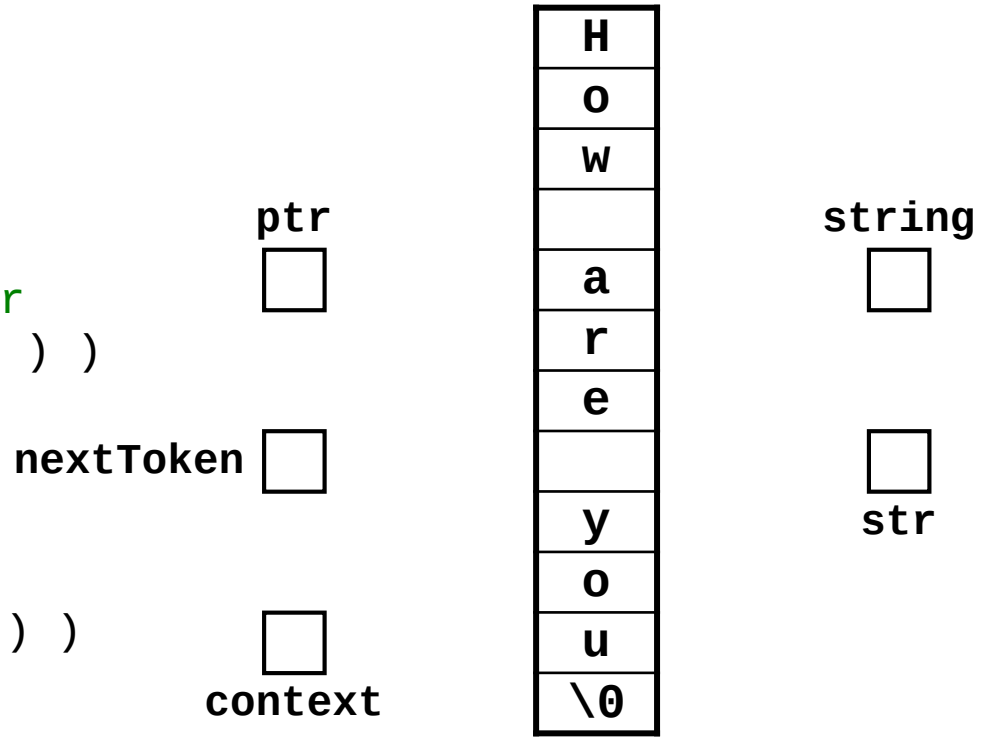
```
char *strtok_s( char *string, const char *control, char **context )
{
  char *it;

  if( string == nullptr )
    it = *context;
  else
    it = string;

  // Find the next nondelimiter
  while( belong( *it, control ) )
    it++;

  string = it;

  // Find the next delimiter
  for( ; *it != '\0'; it++ )
    if( belong( *it, control ) )
    {
      *(it++) = '\0';
      break;
    }

  *context = it;

  if( string == it )
    return nullptr;
  else
    return string;
}
```




```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != nullptr )
{
  cout << ptr << '\n';
  ptr = strtok_s( nullptr, " ", &nextToken );
}
```

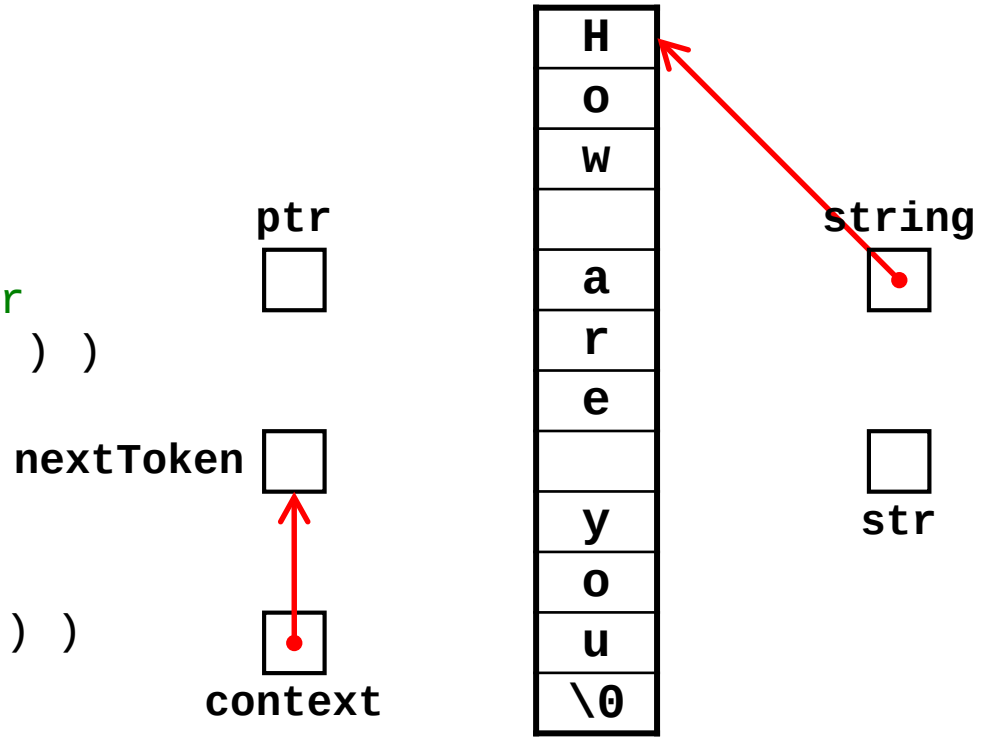
```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next delimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next nondelimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



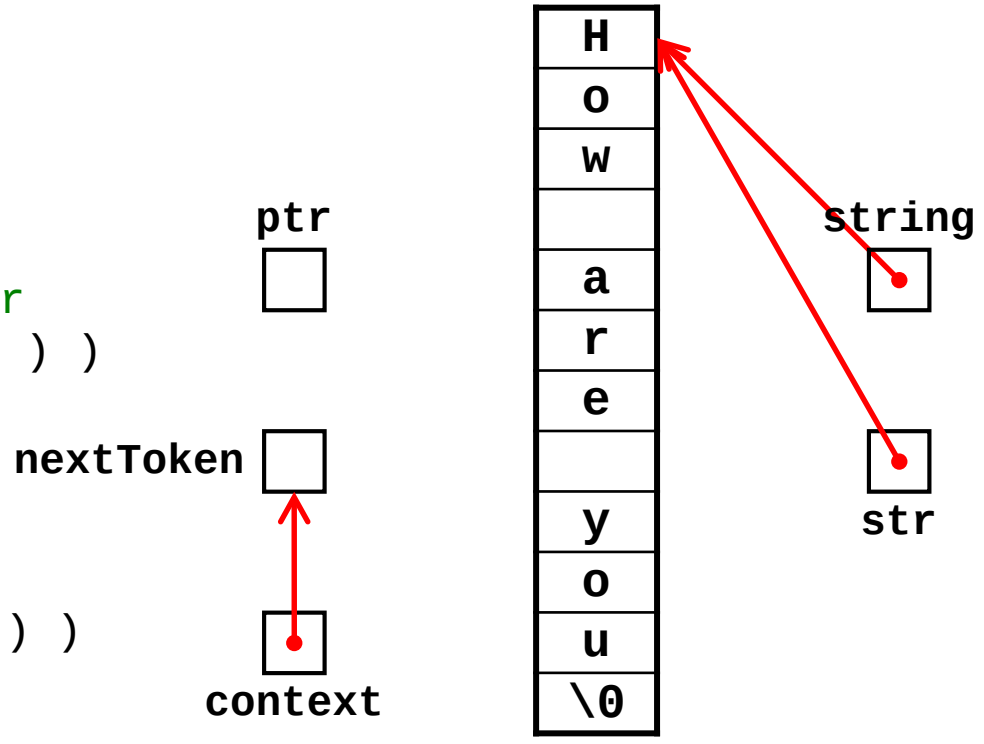

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



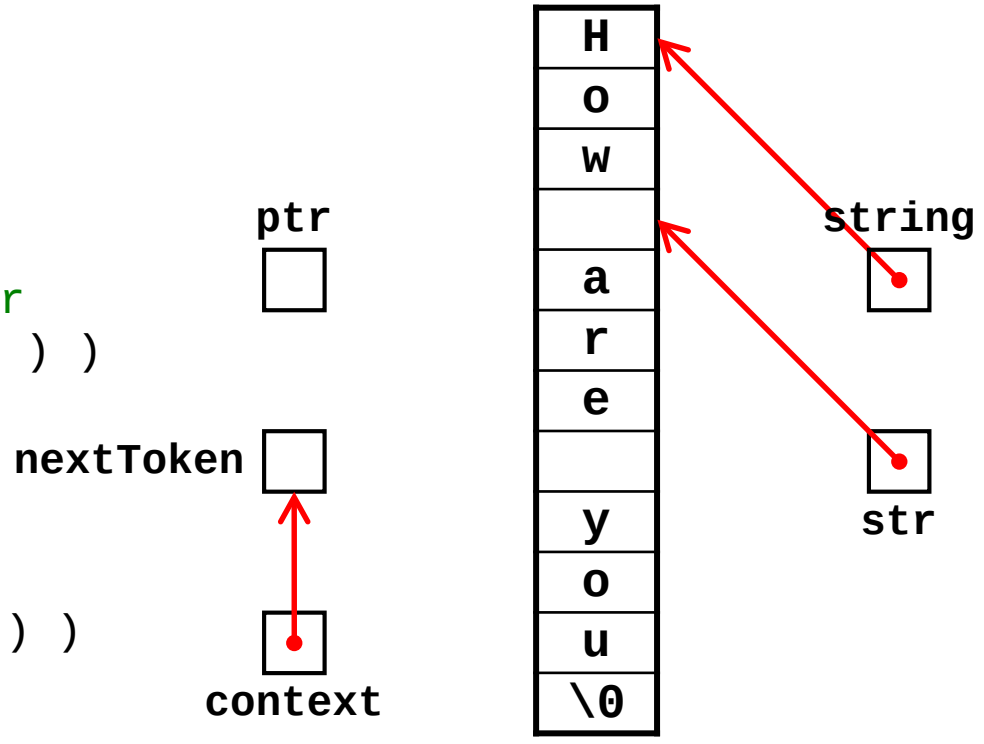

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



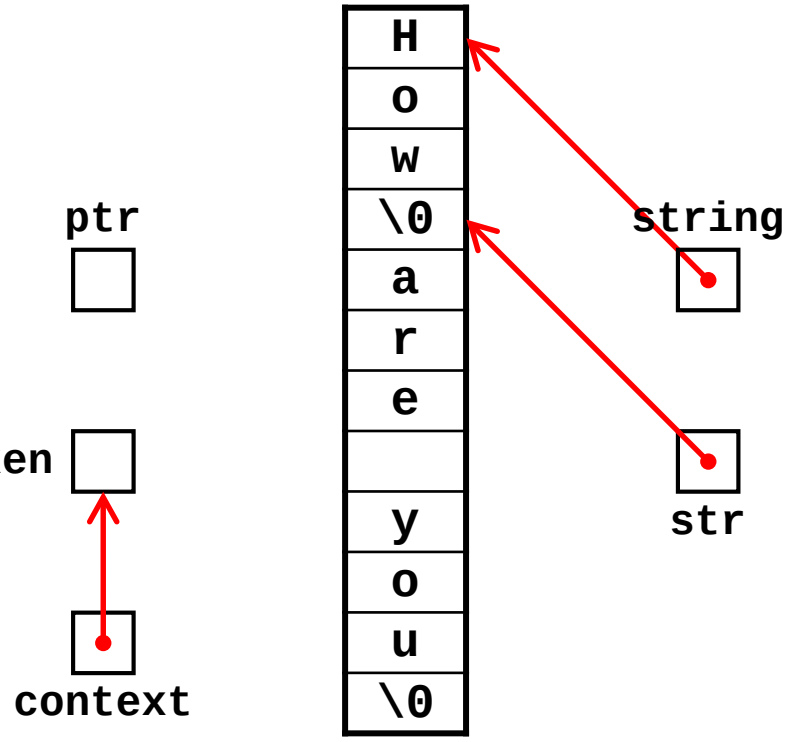

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



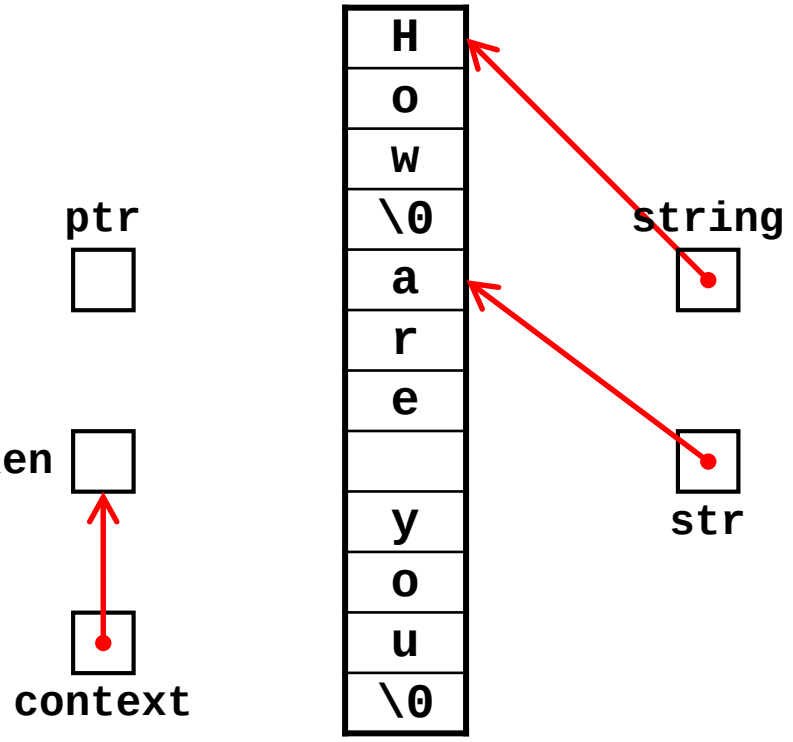

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



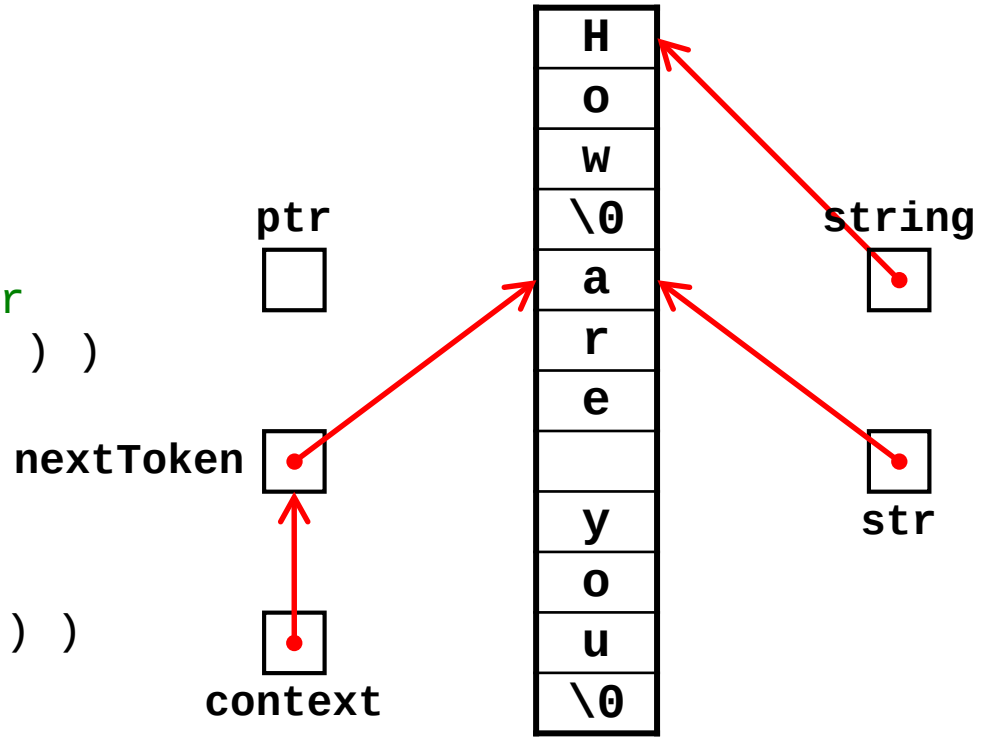

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



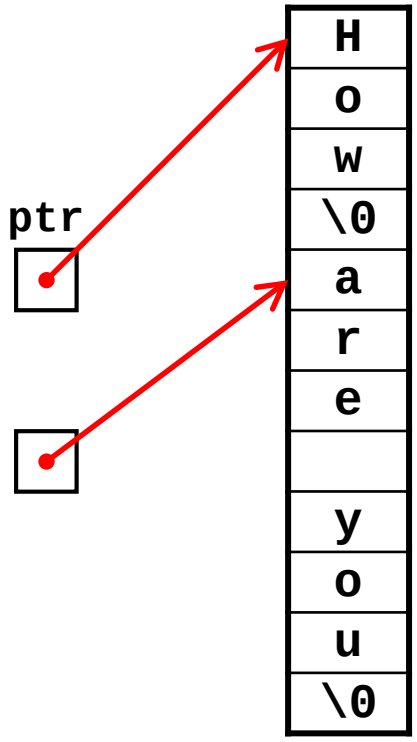

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



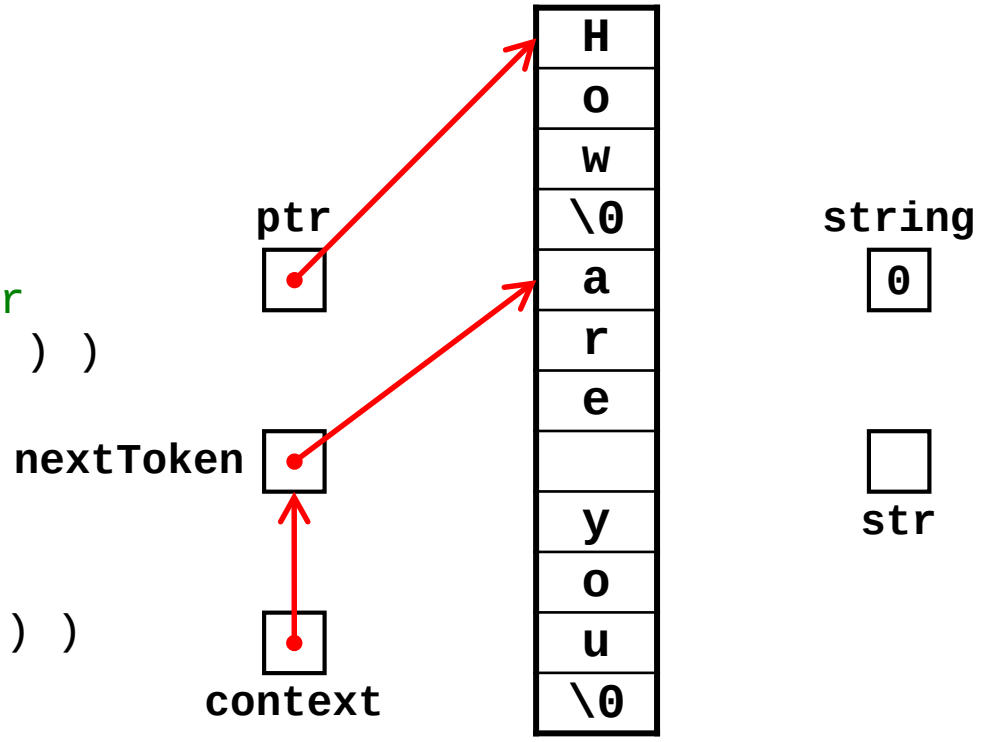

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



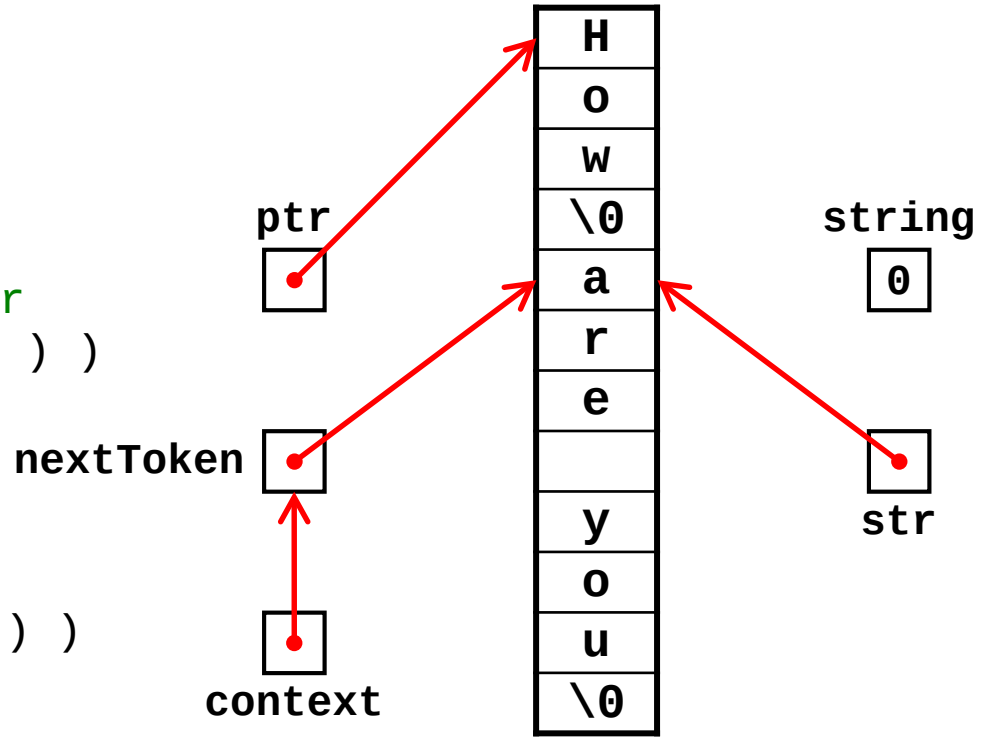

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



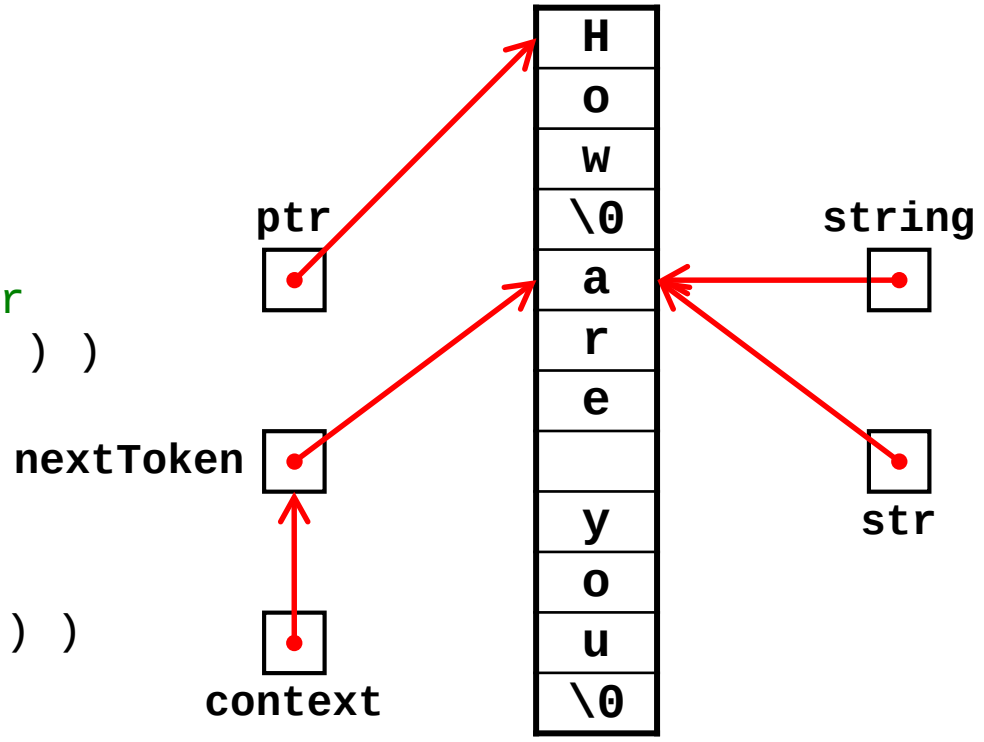

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



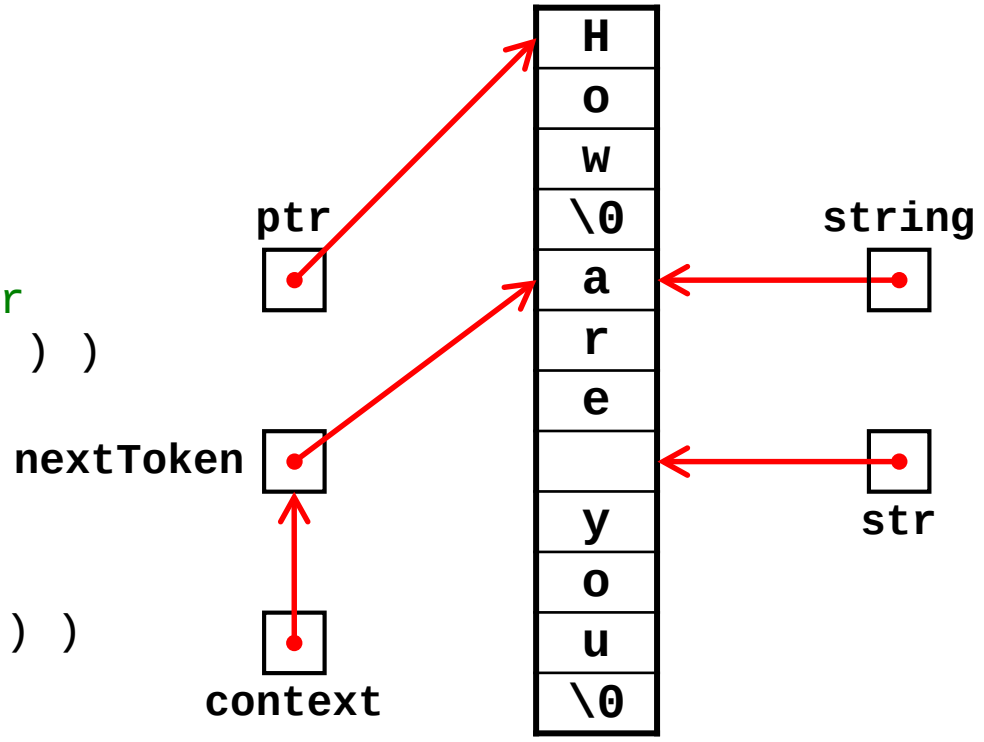

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



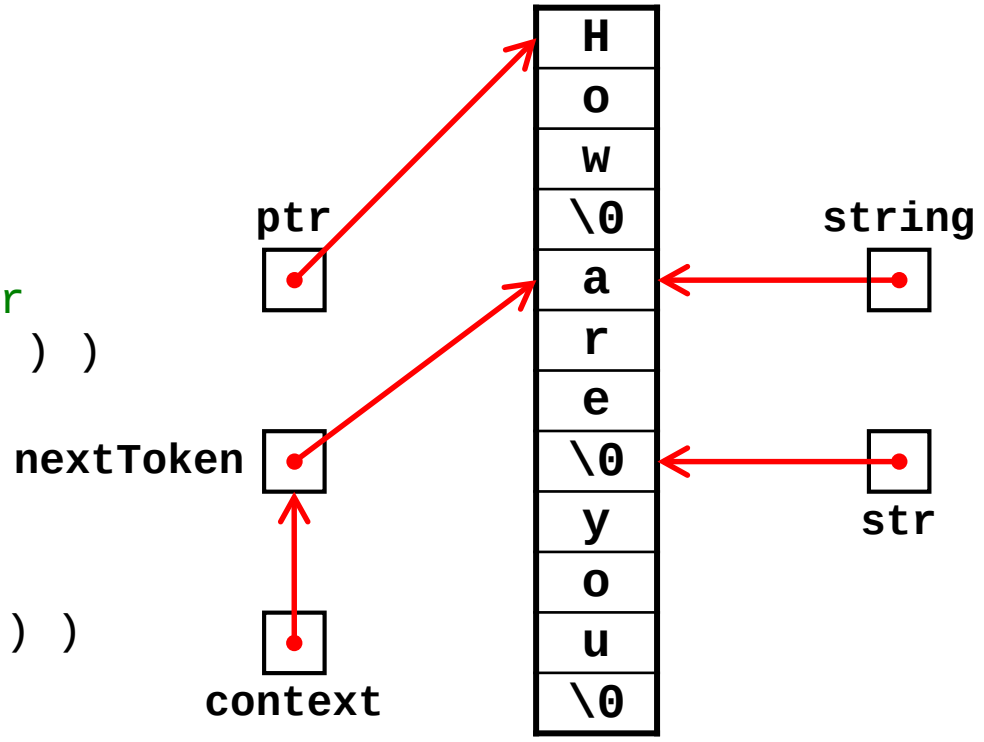

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



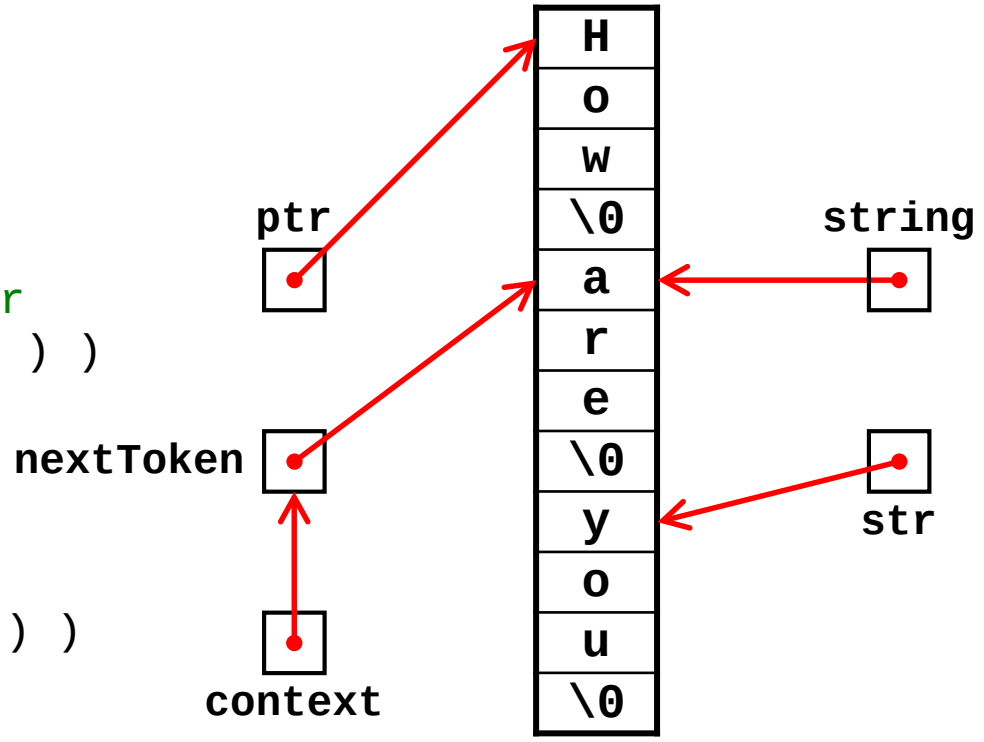

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



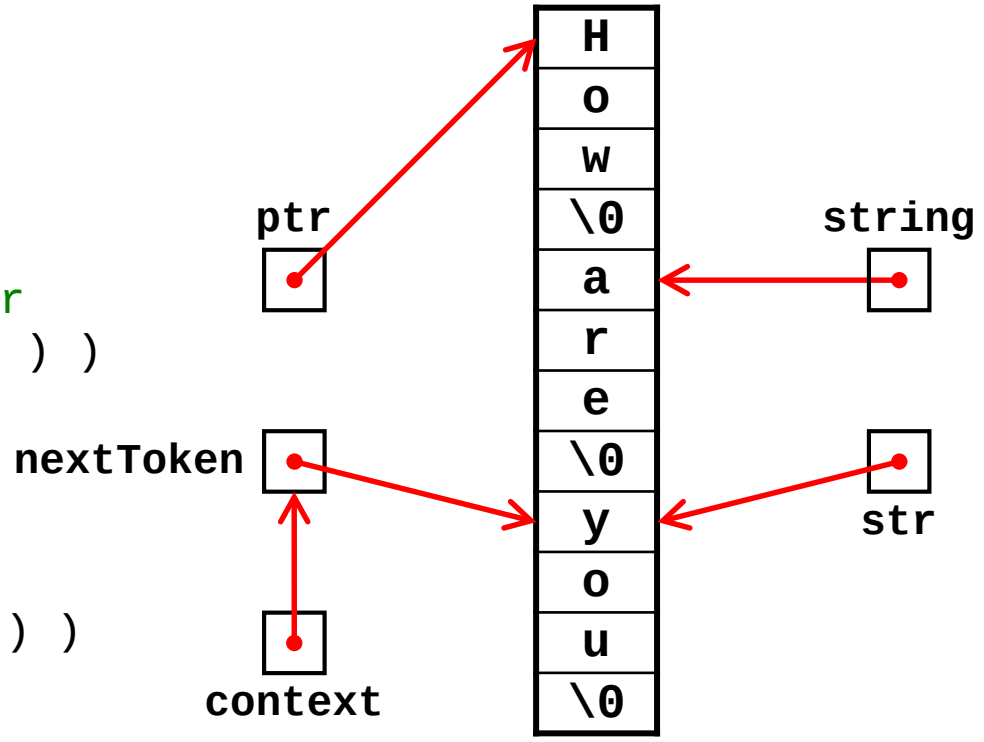

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



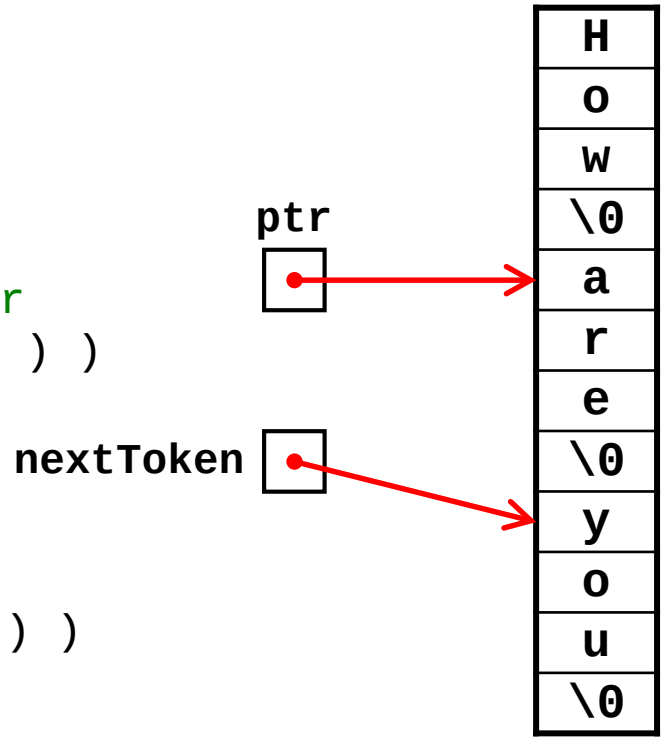

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



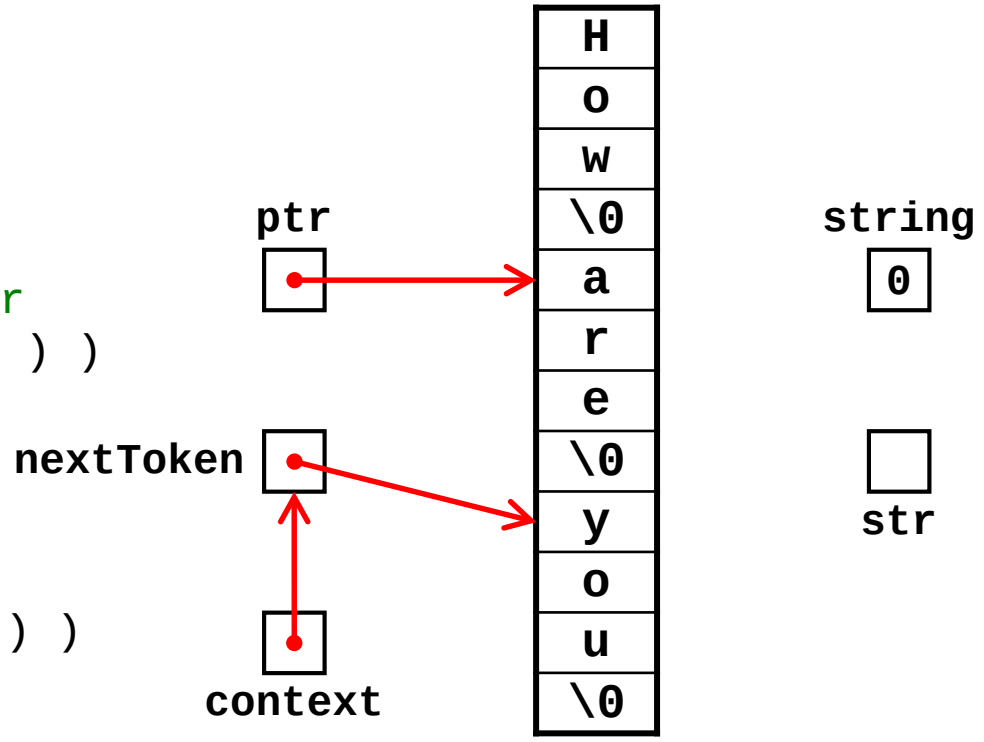

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



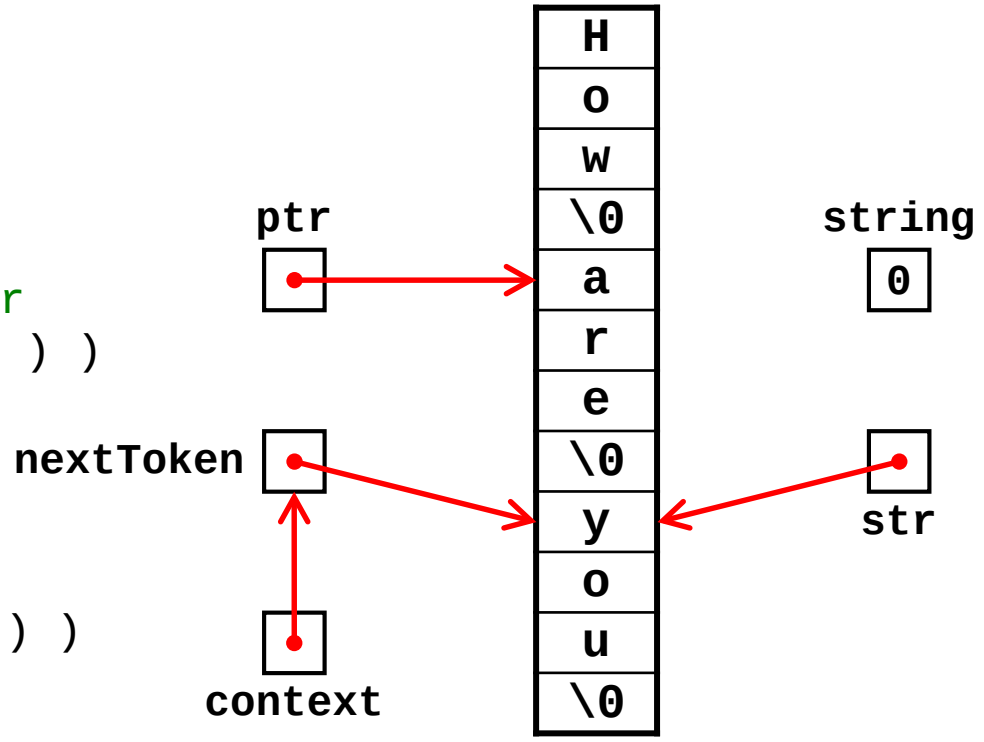

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



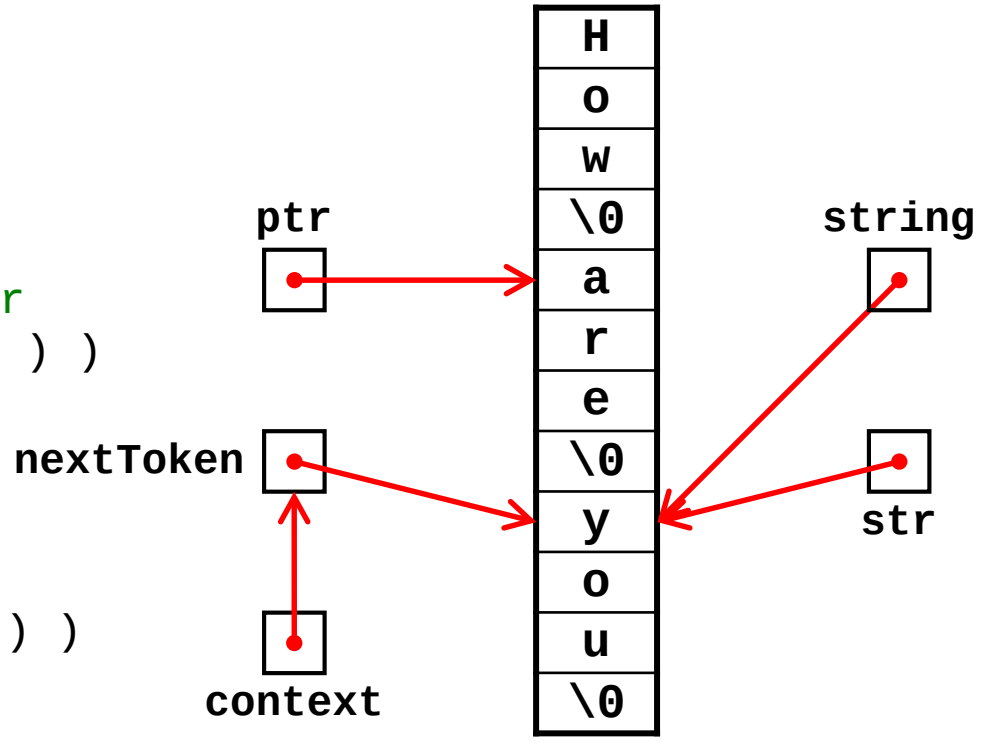

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



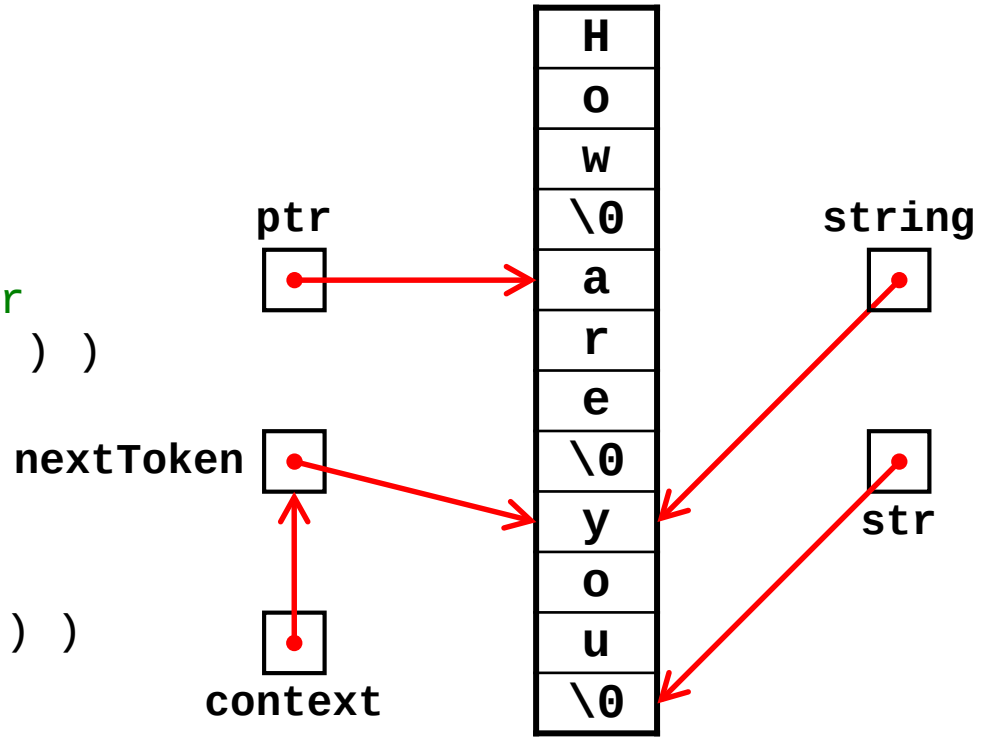

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



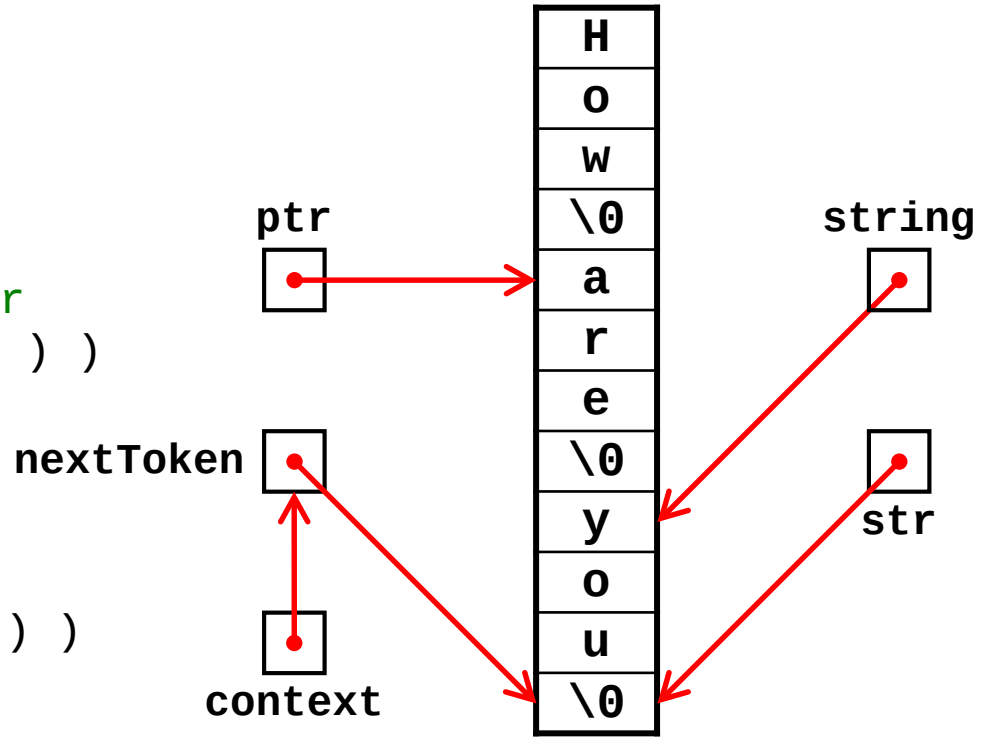

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



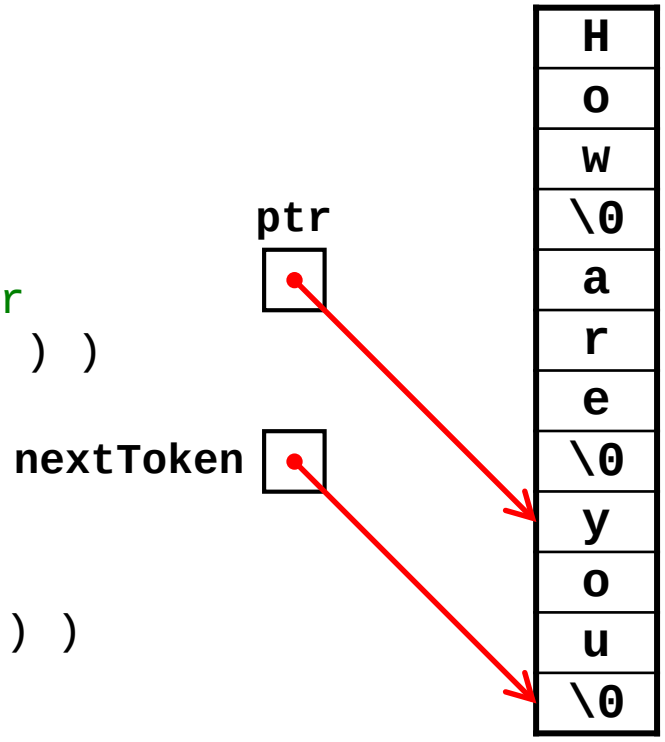

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



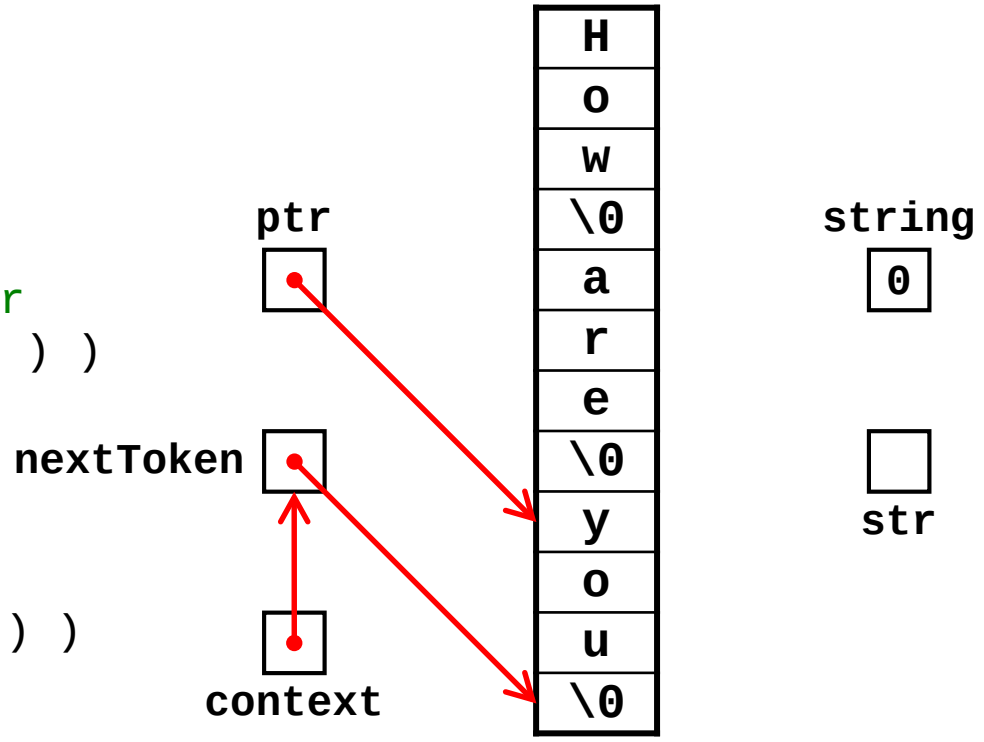

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



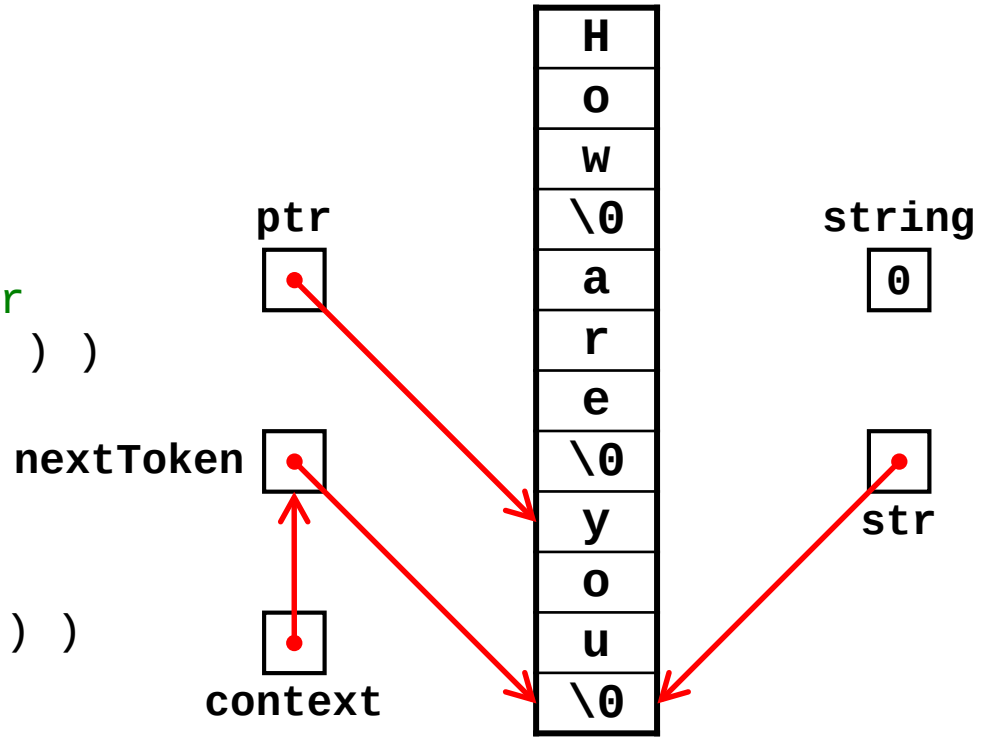

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



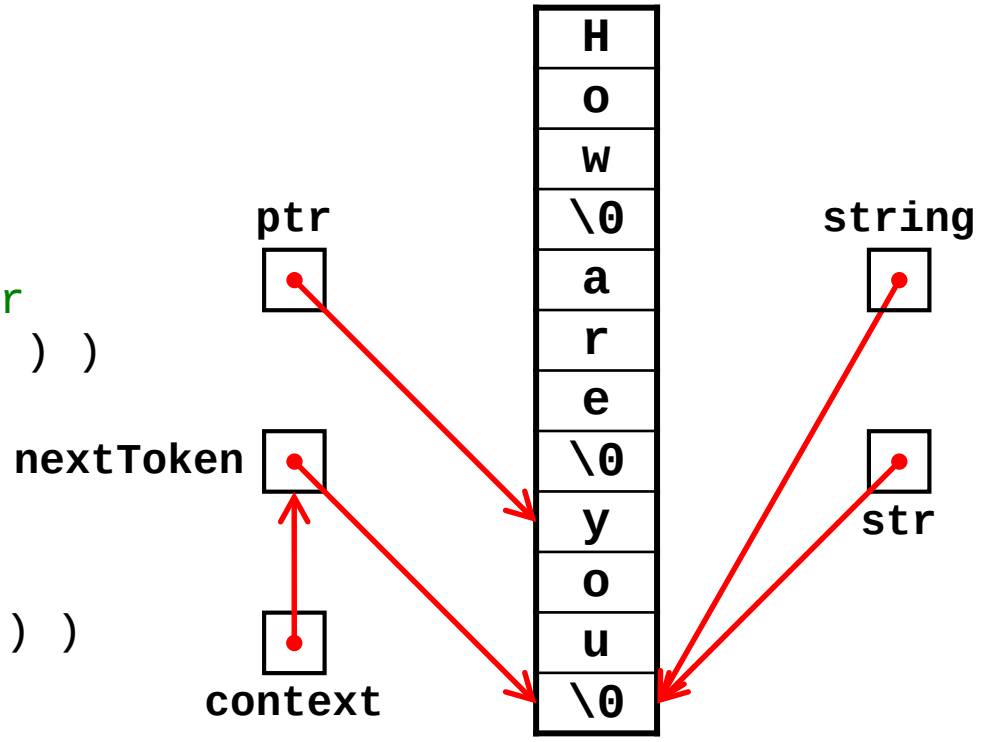

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



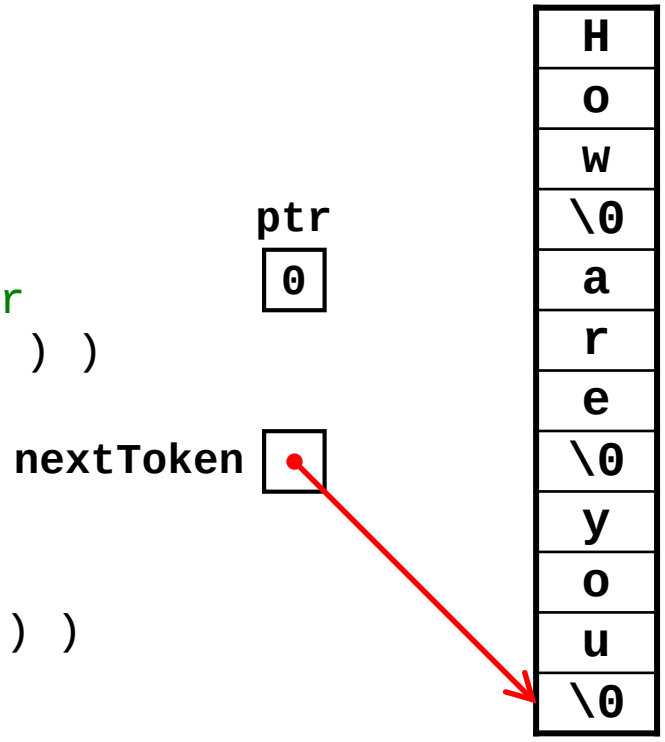

```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```




```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```




```
ptr = strtok_s( string, " ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, " ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```

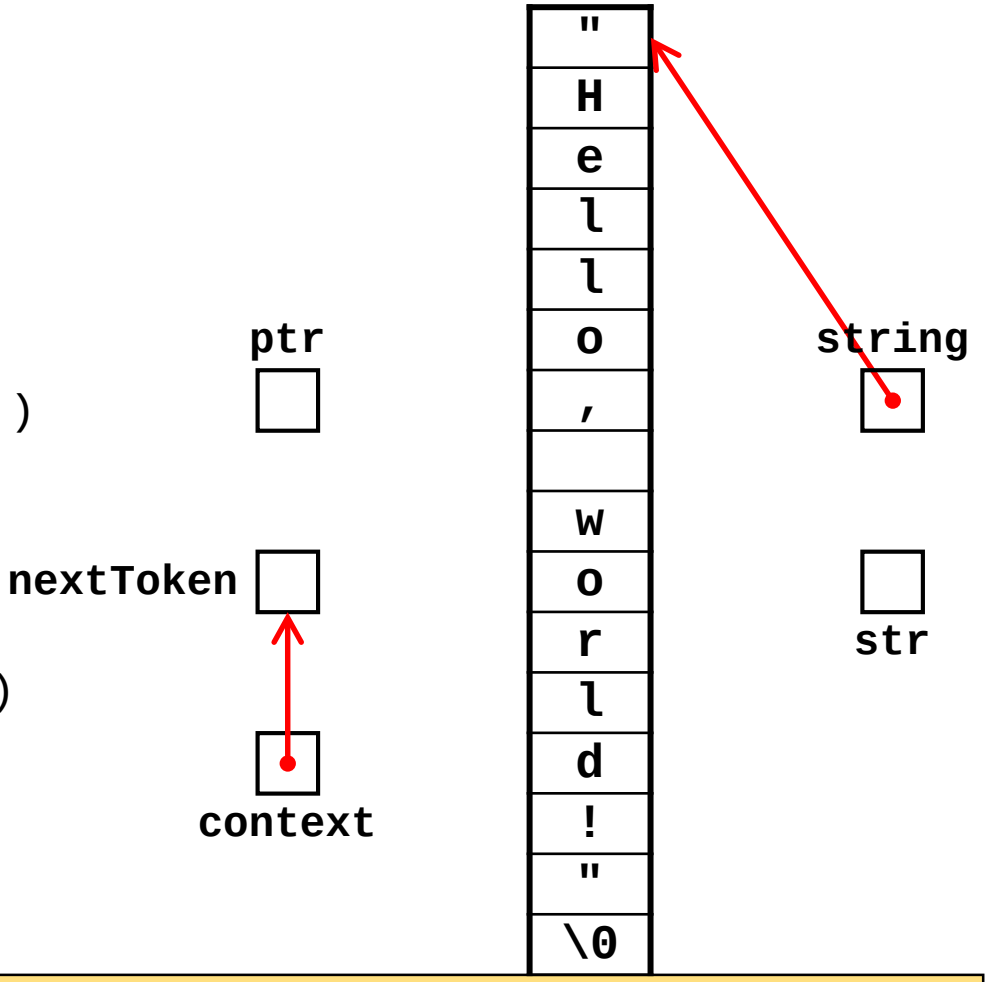
ptr

nextToken

"
H
e
l
l
o
,
w
o
r
l
d
!
"
\0

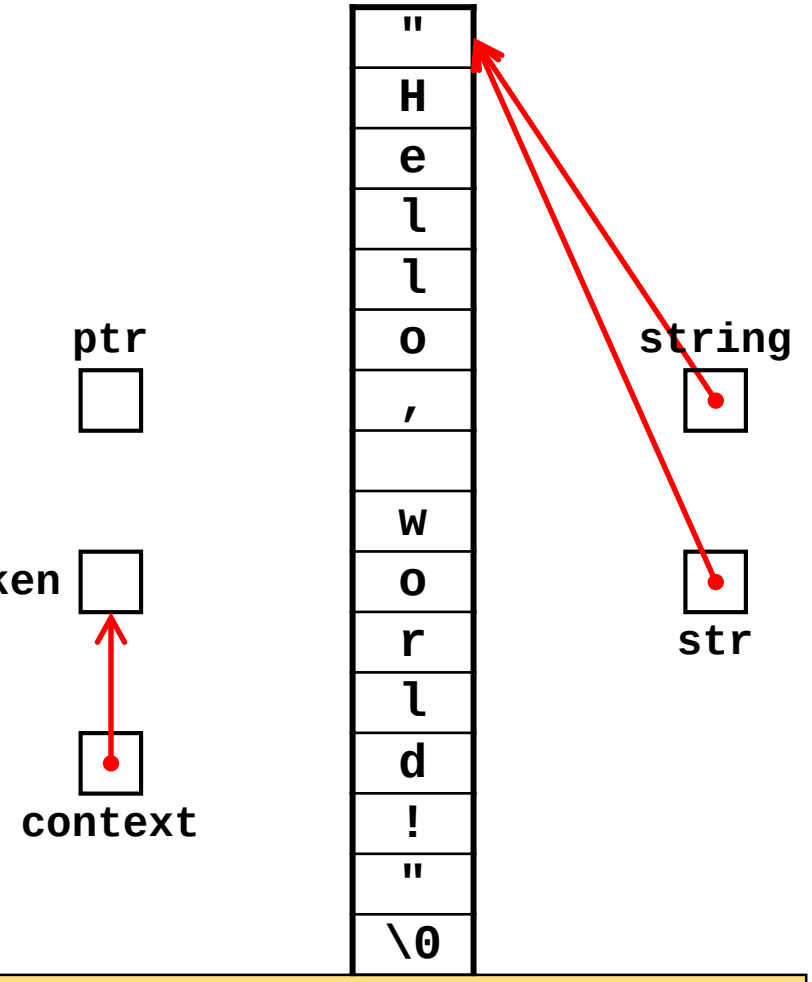
```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



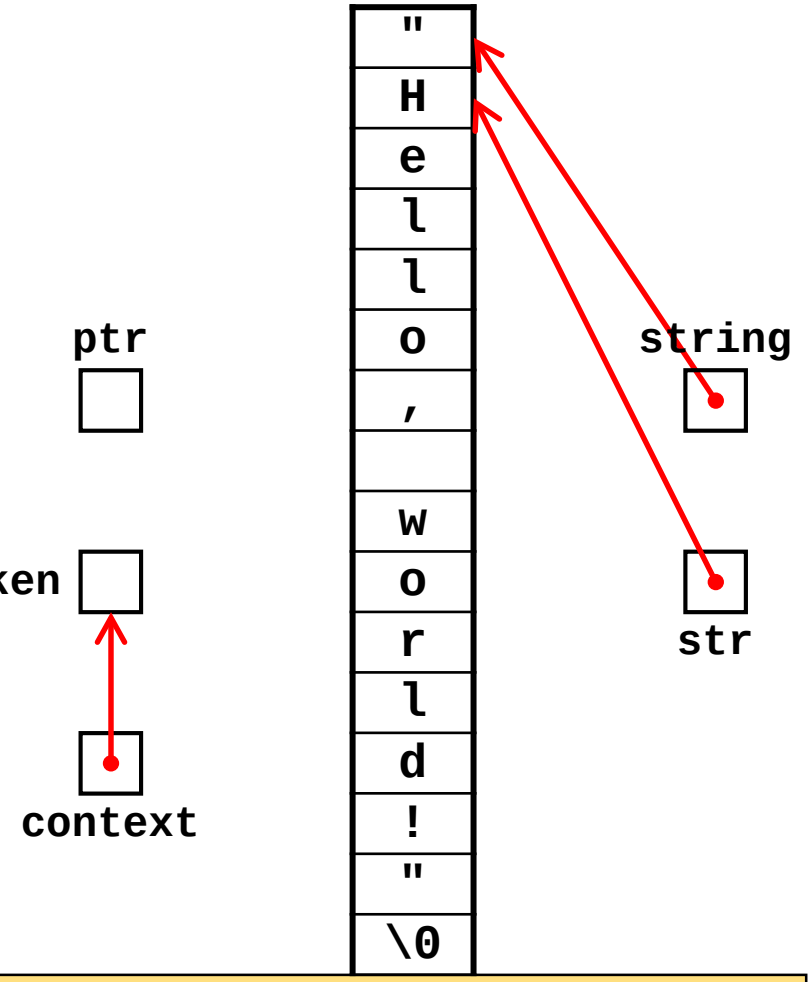

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



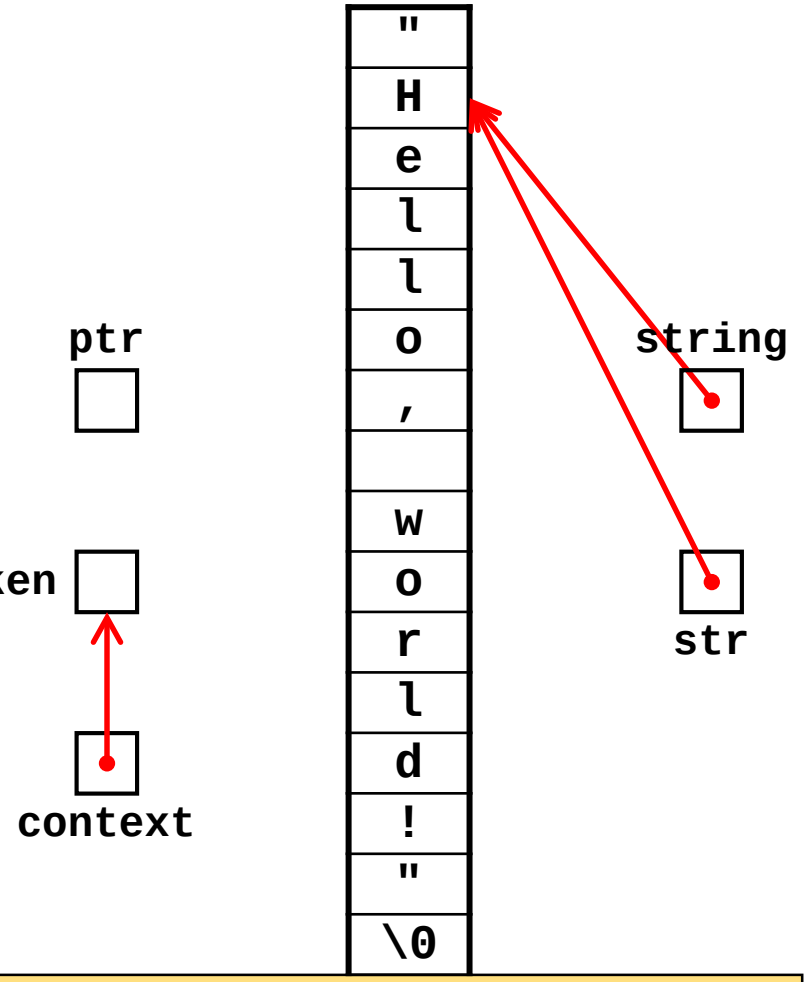

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



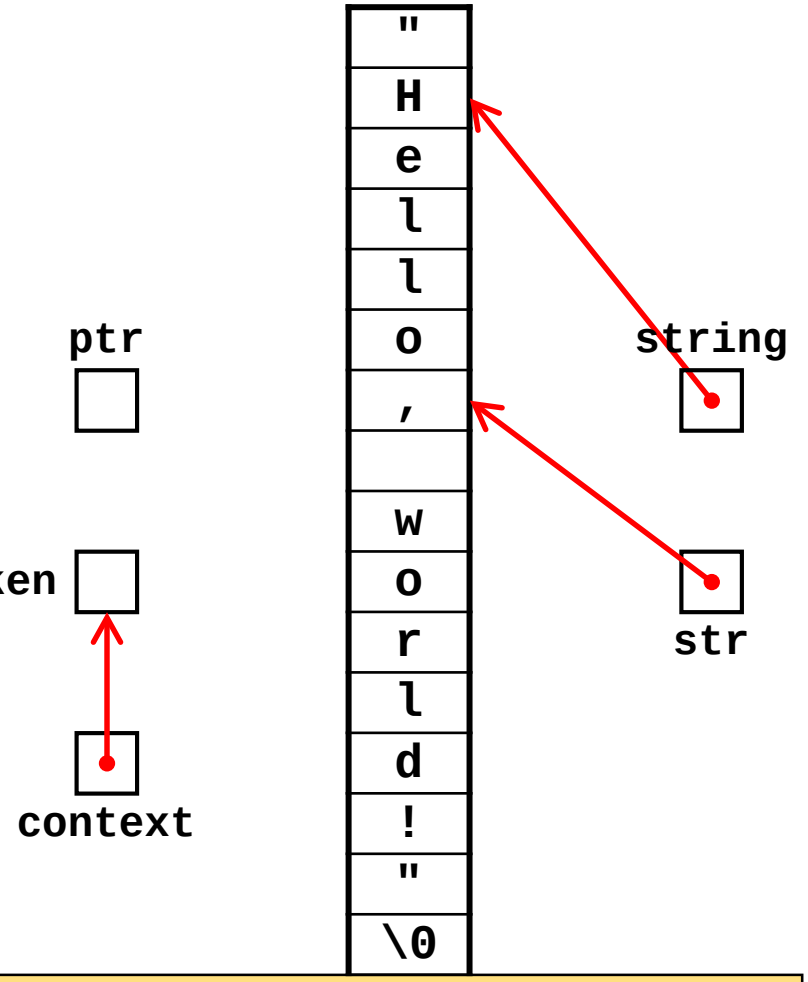

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



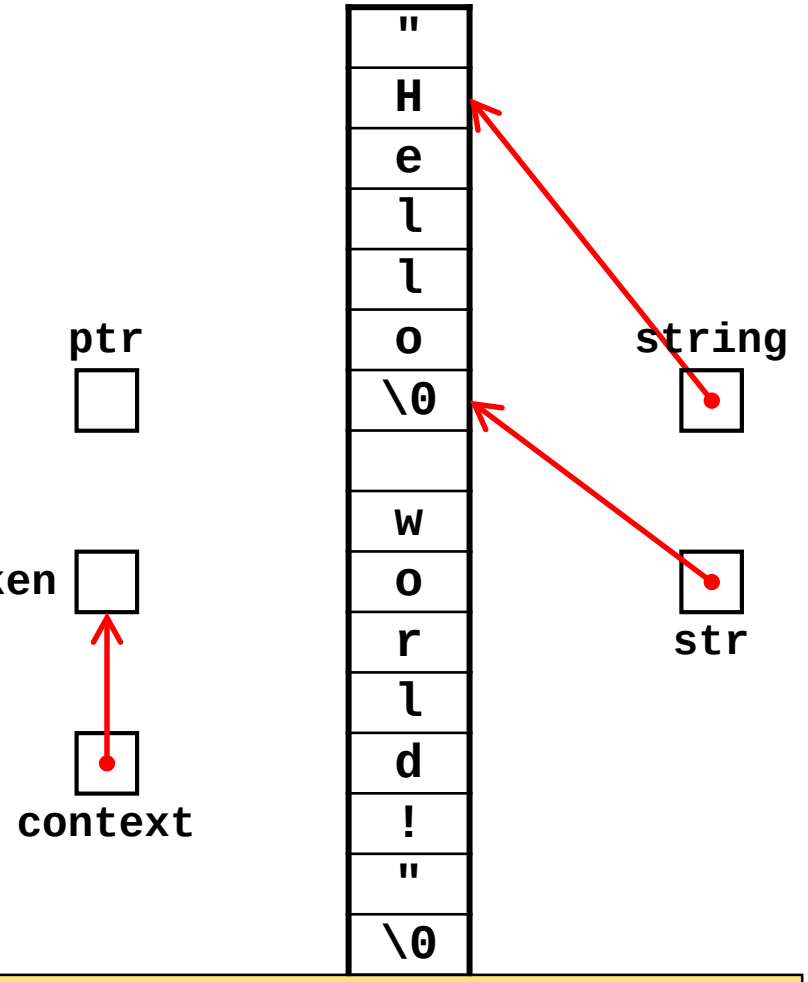

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



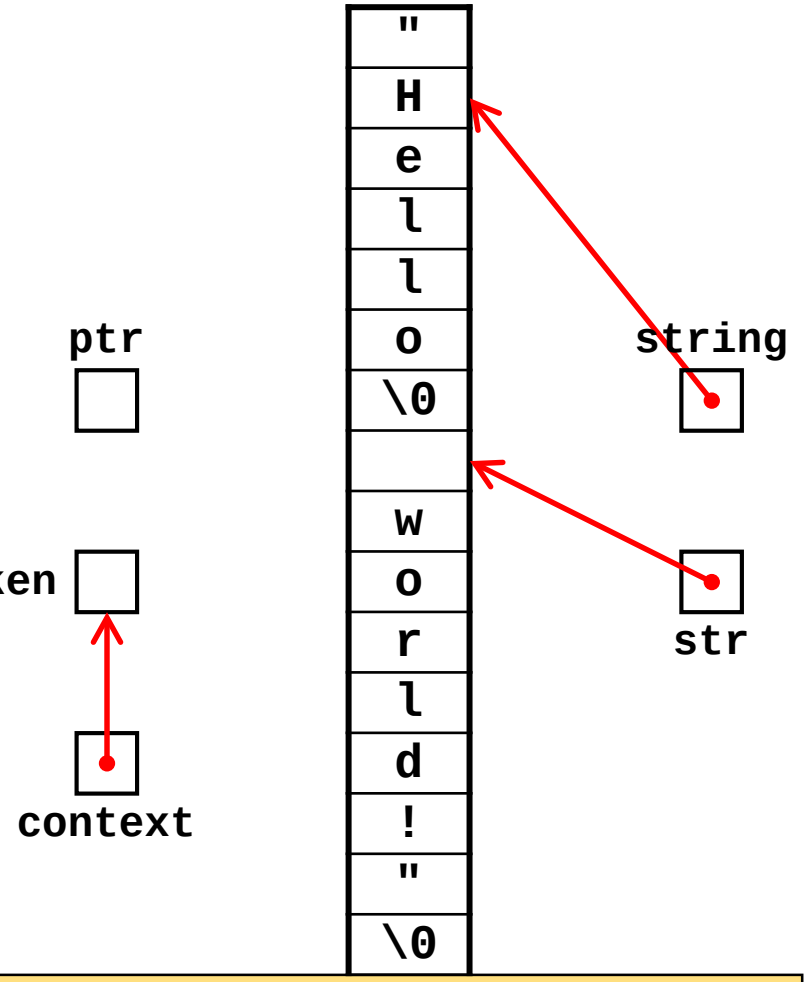

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



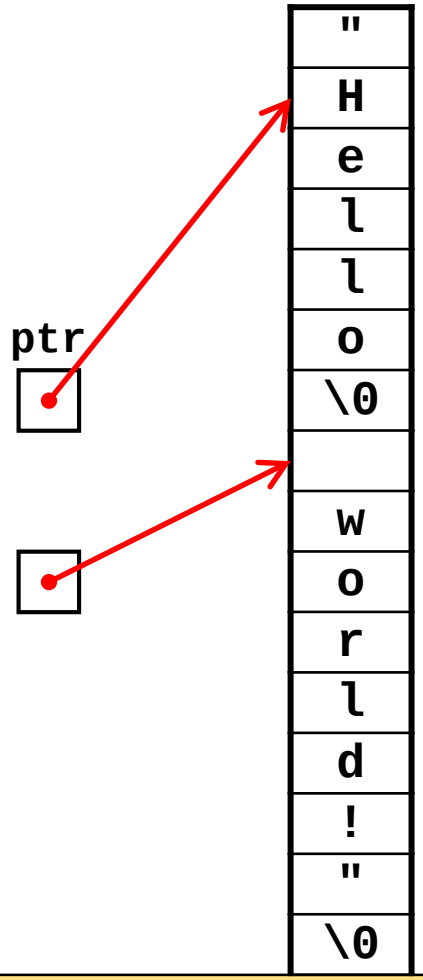

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



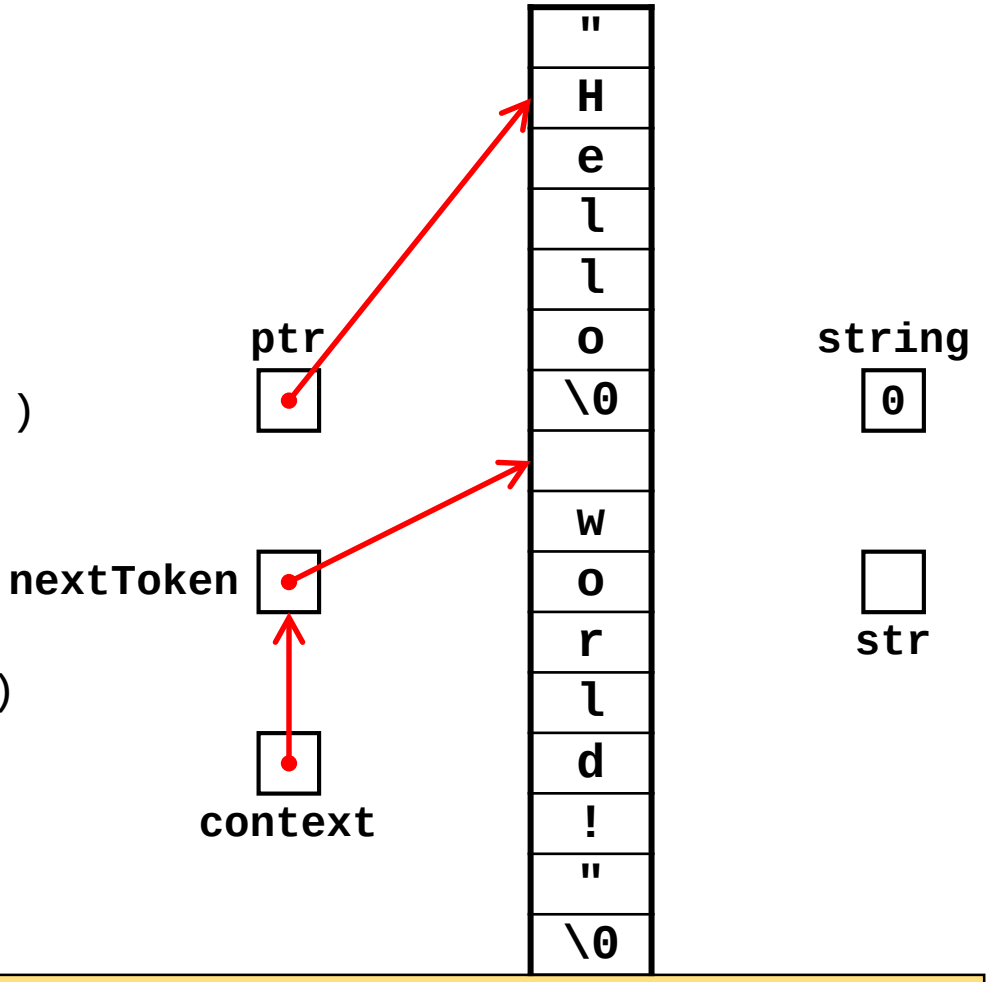

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



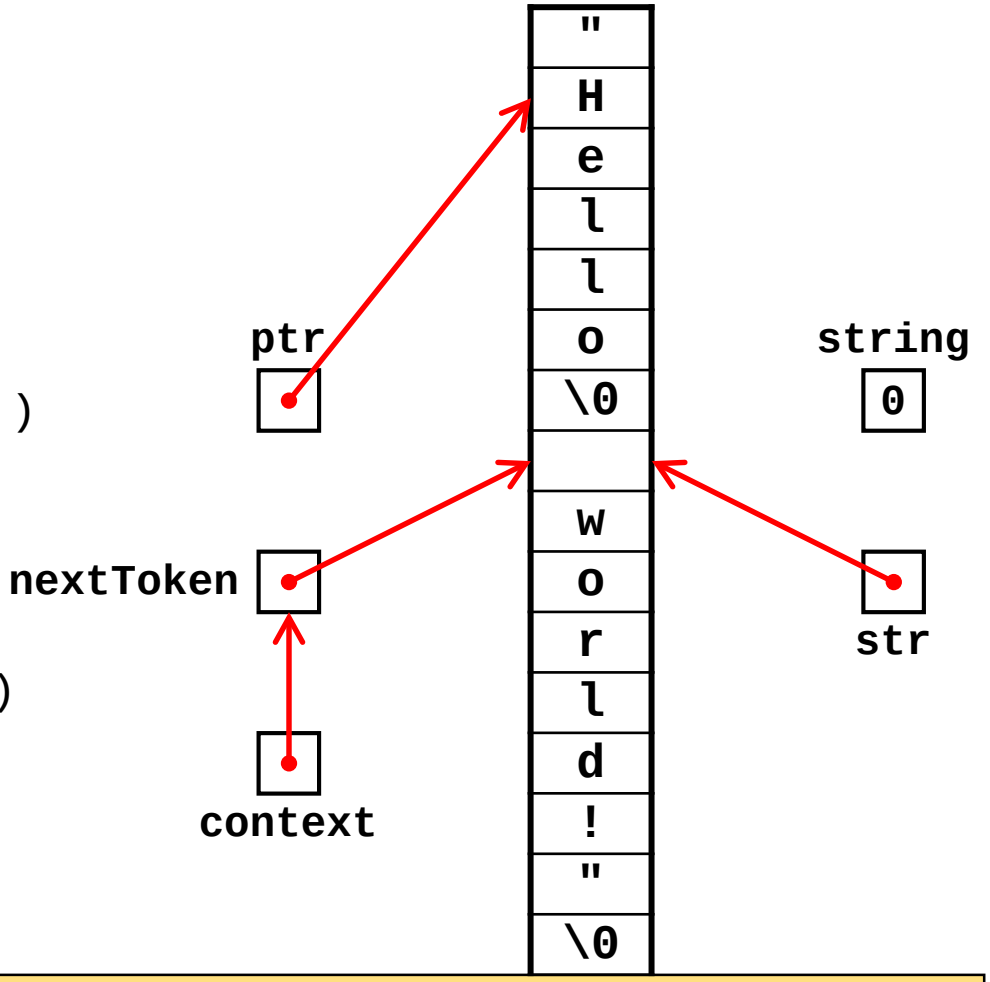

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



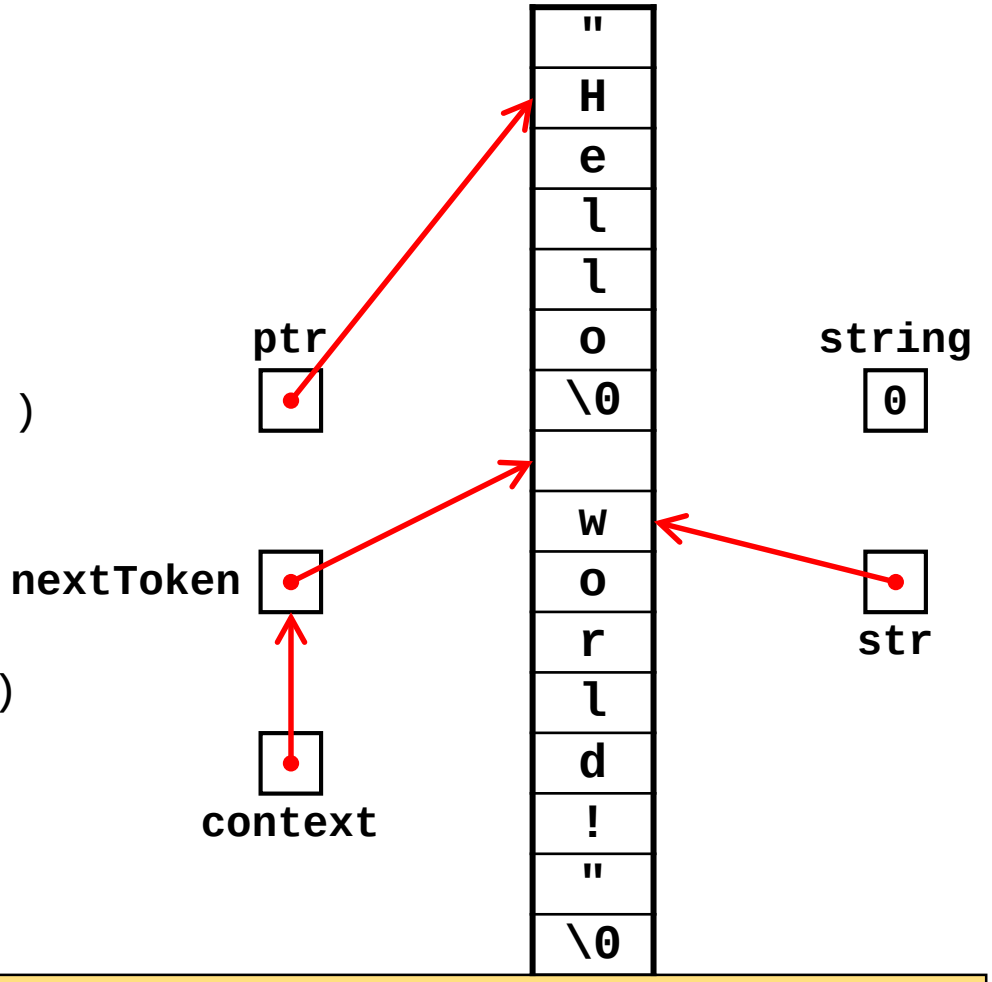

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



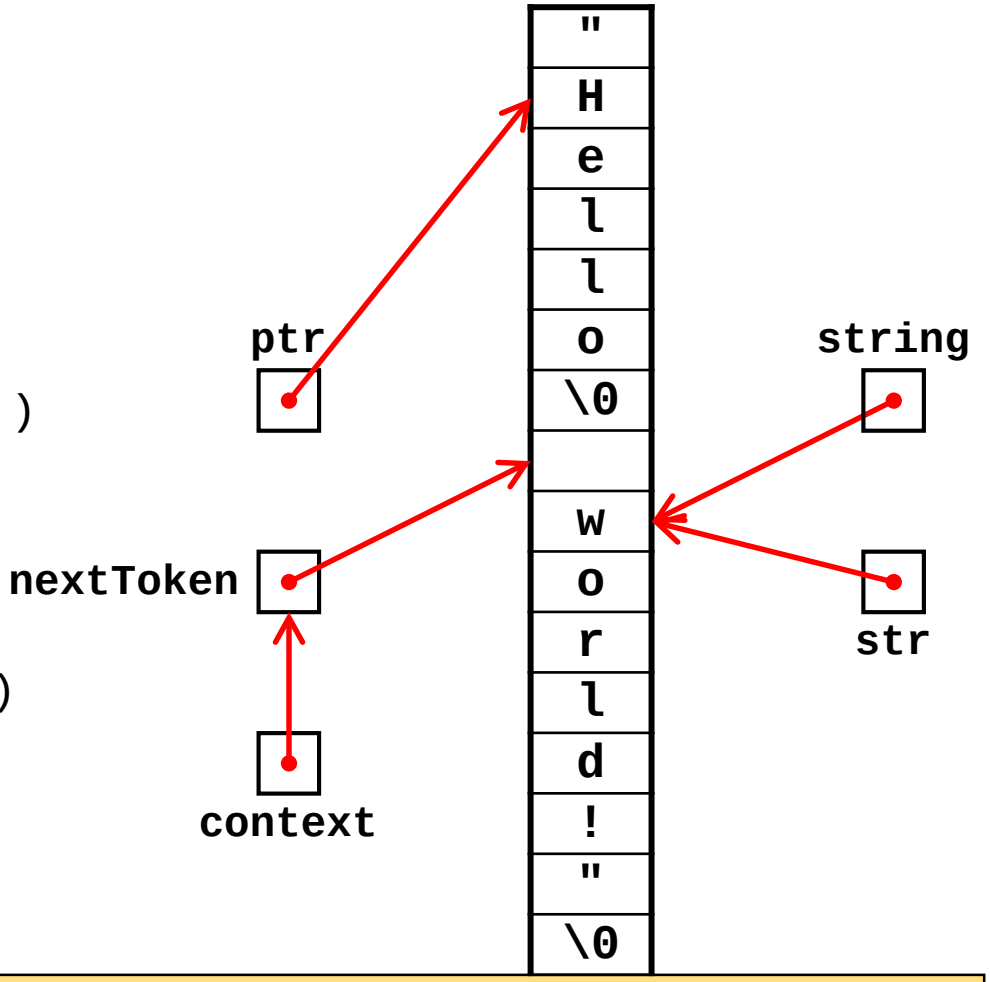

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



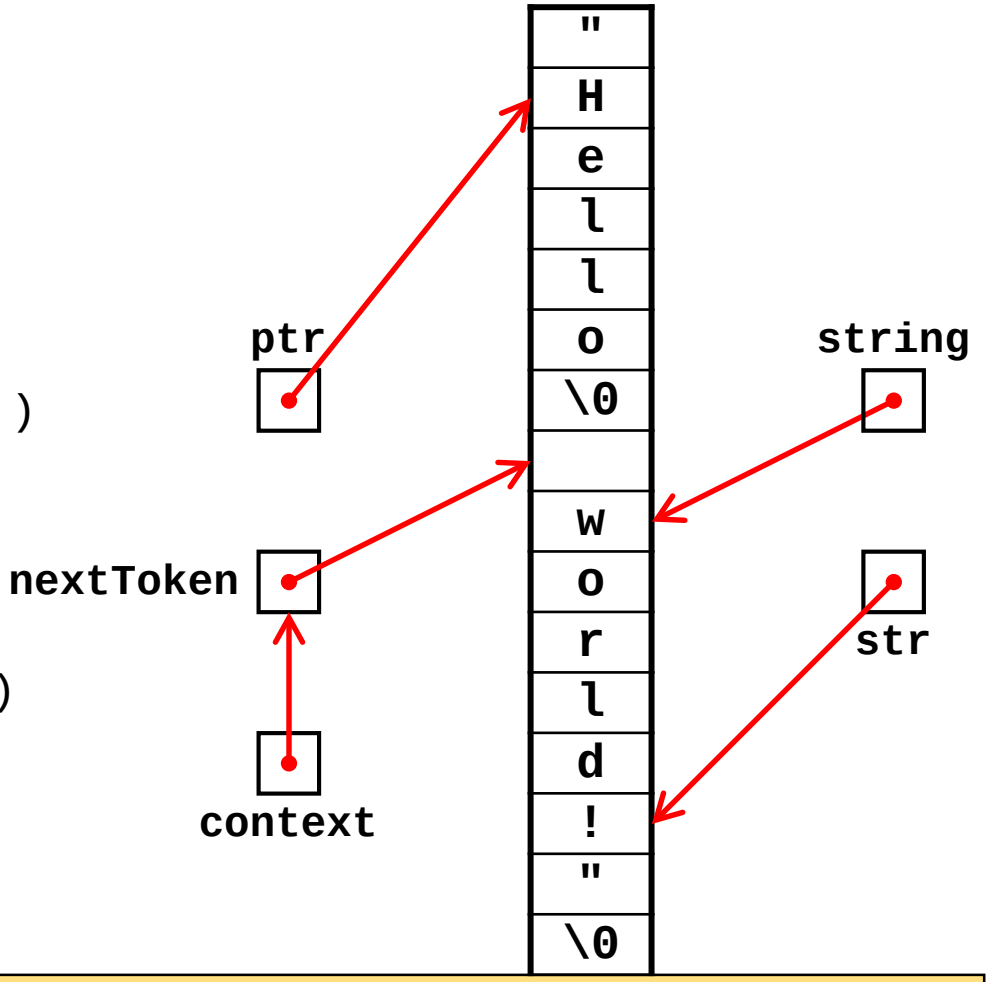

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



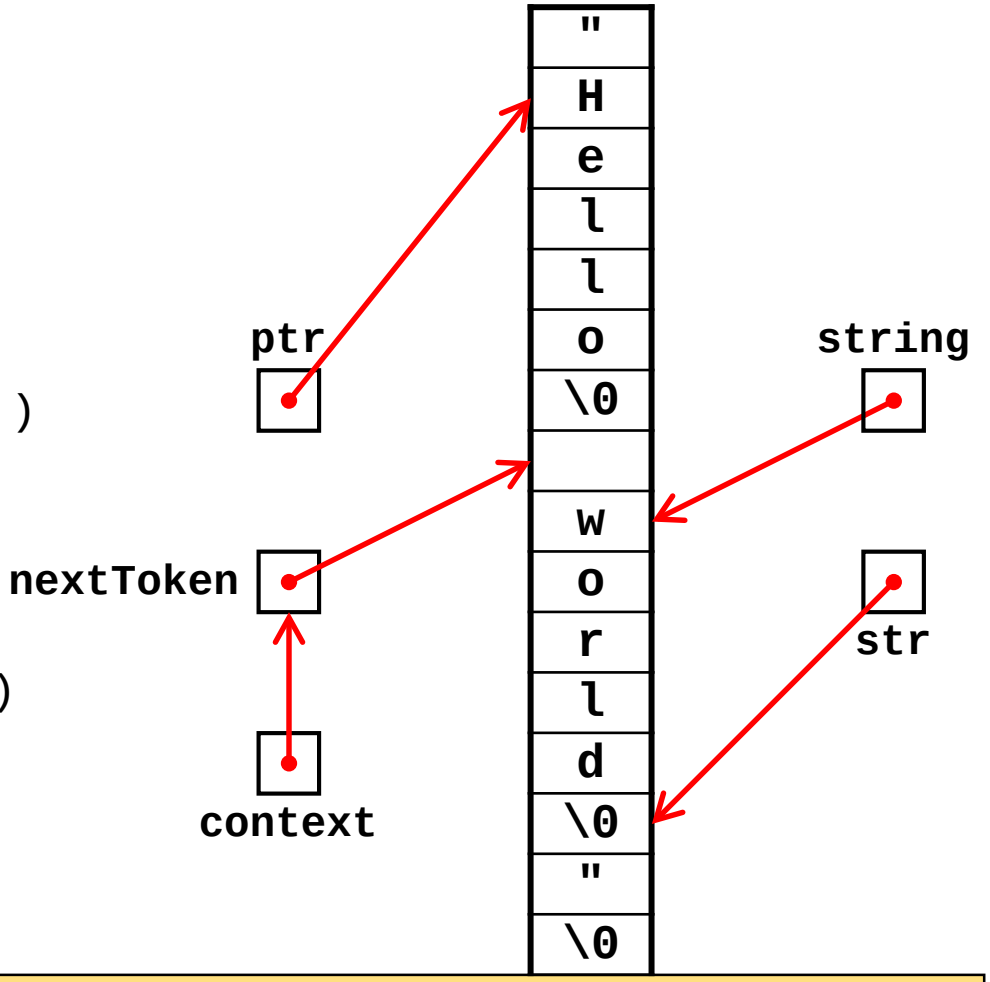

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



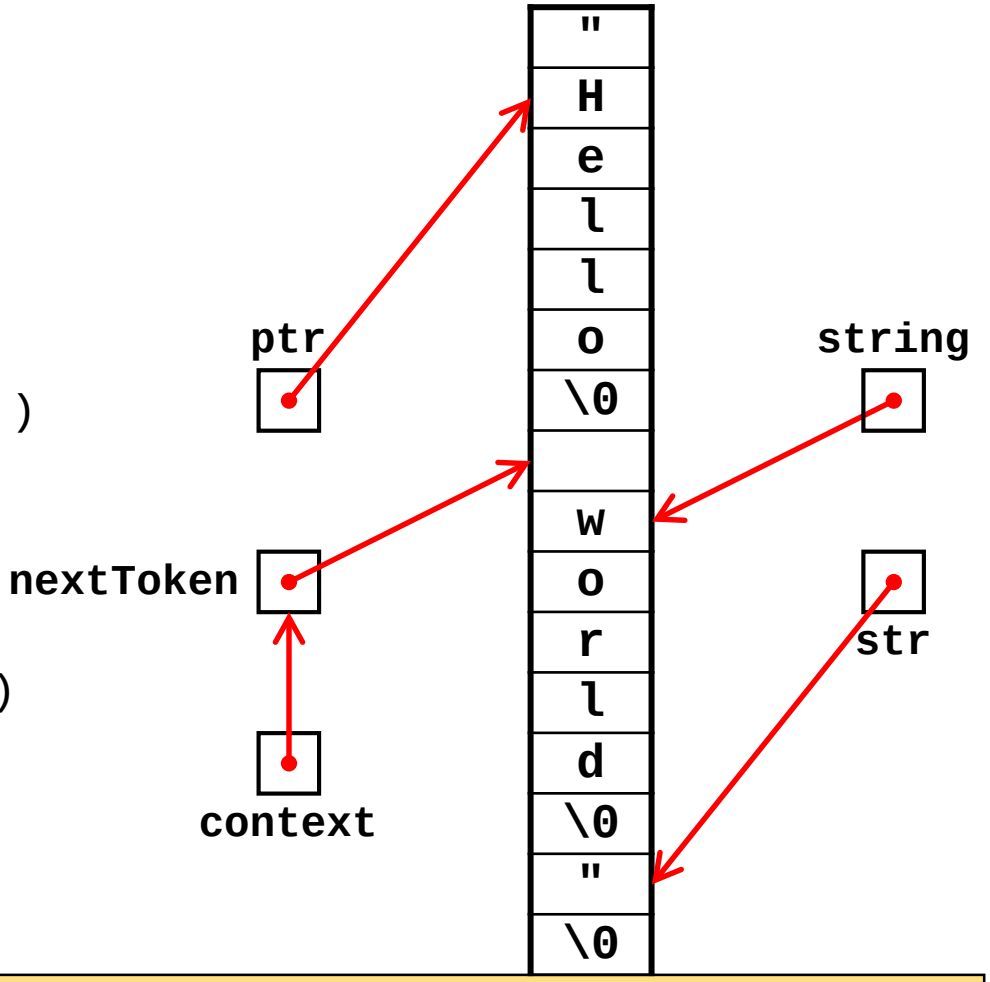

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



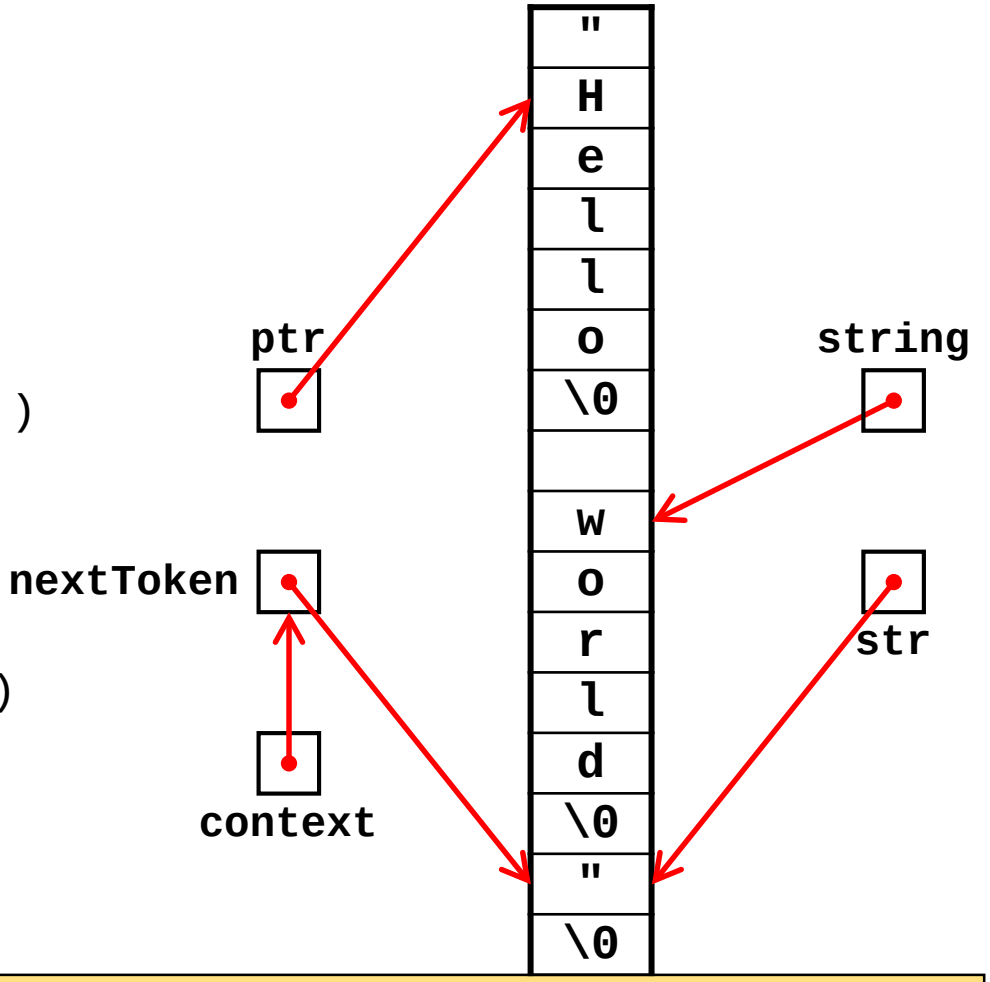

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



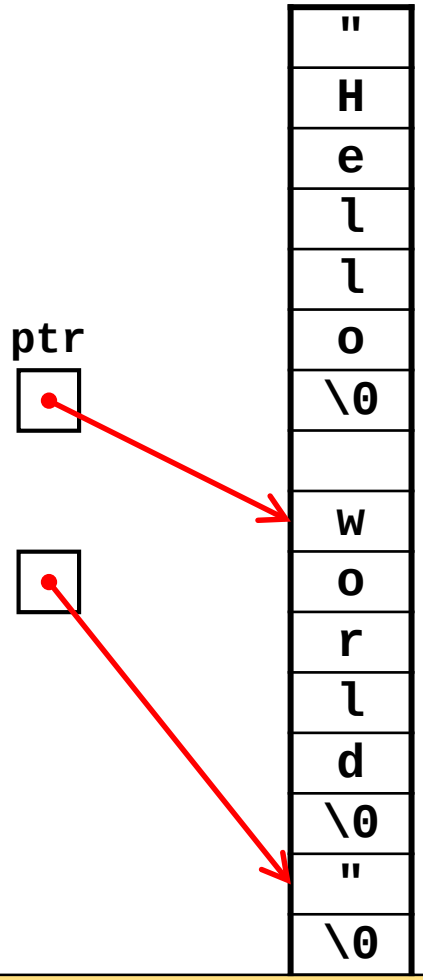

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



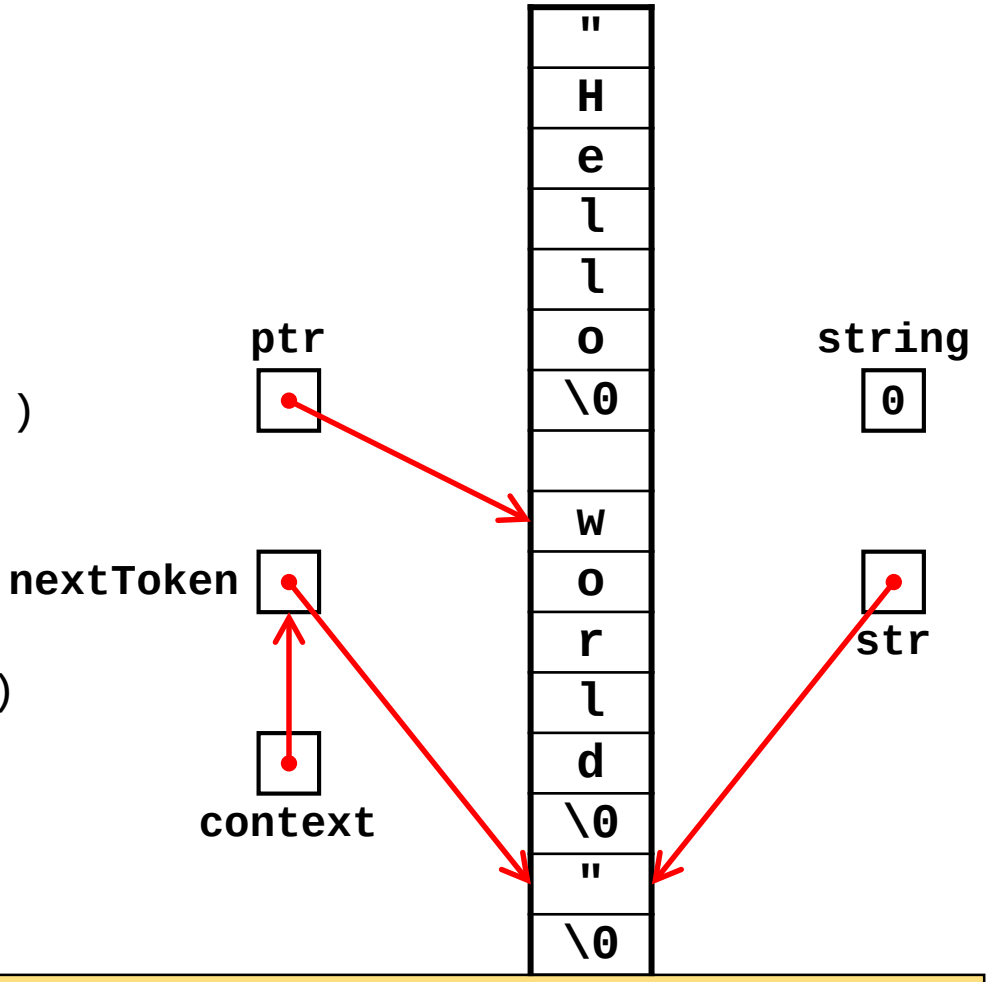

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



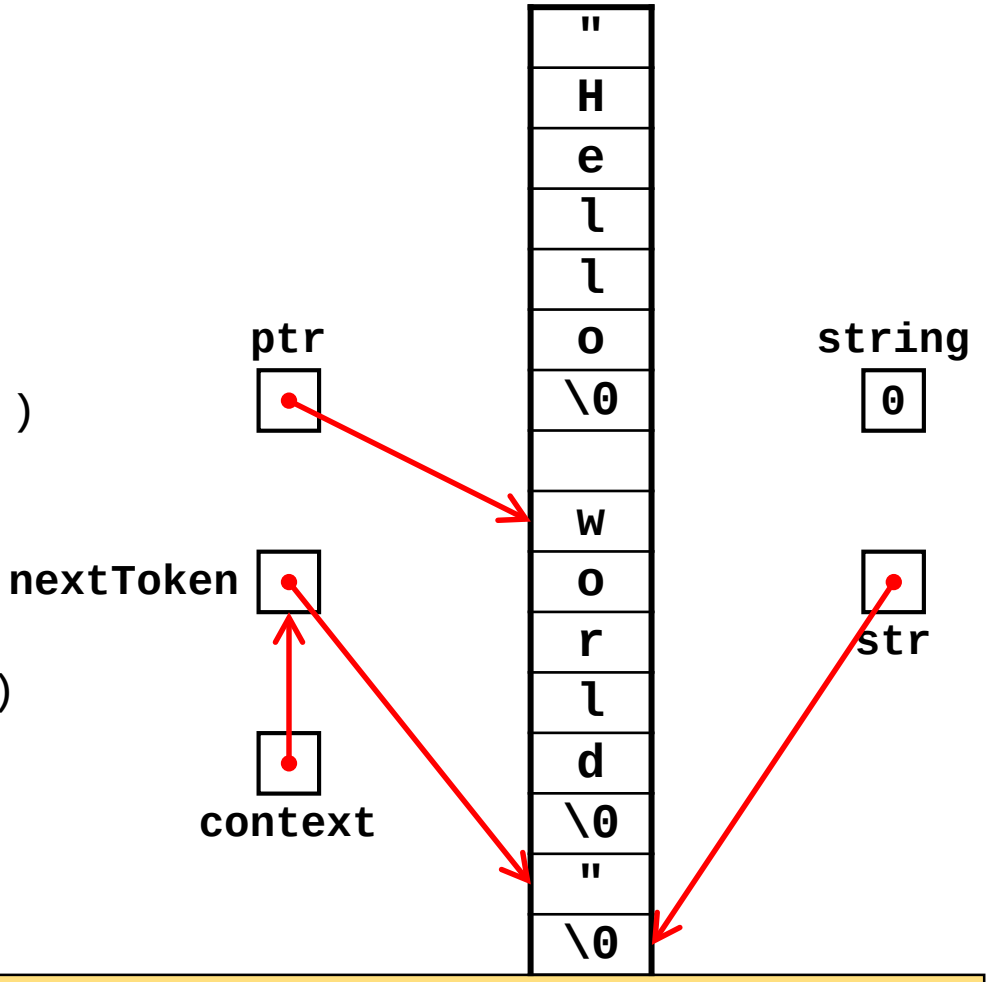

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



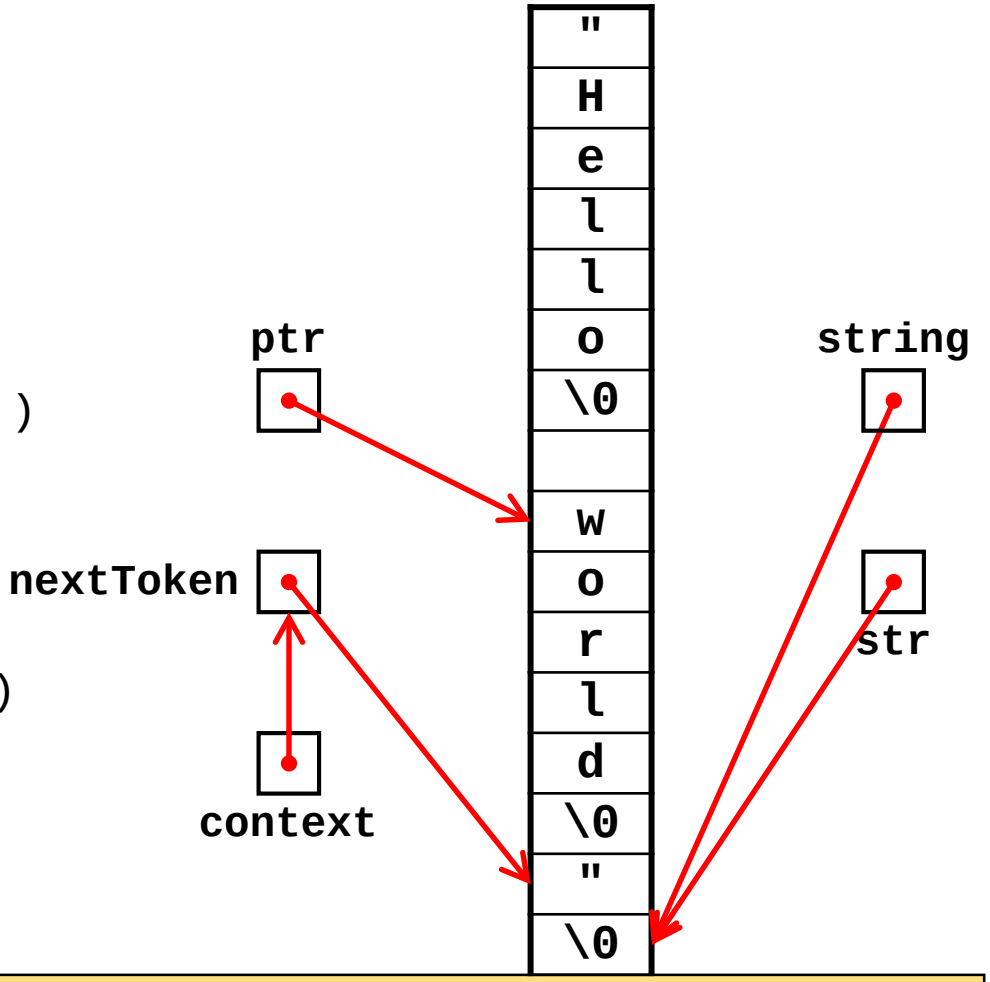

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



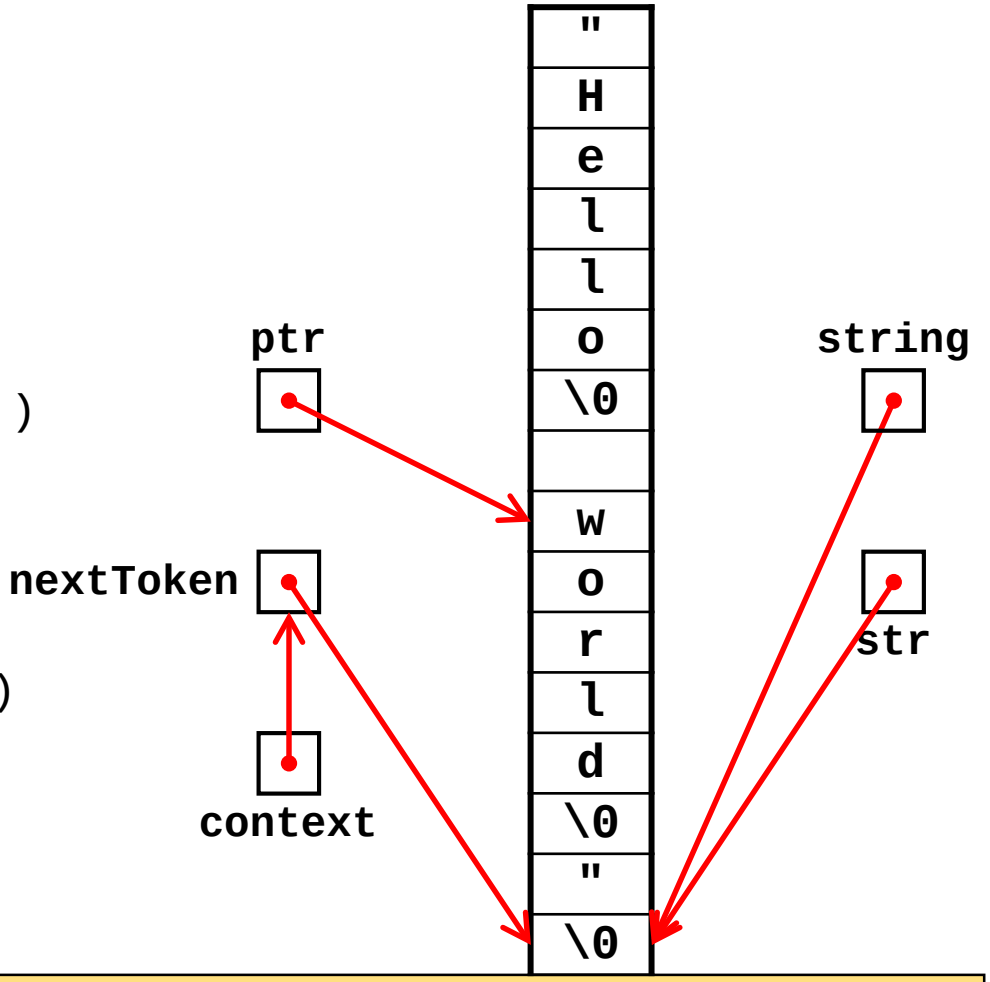

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```



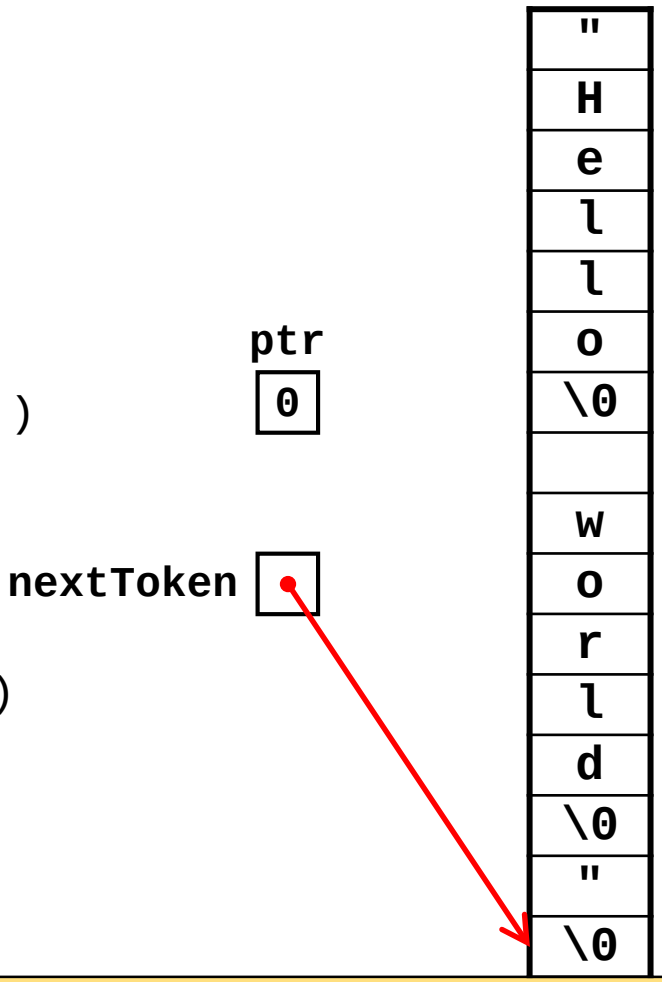

```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```




```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
char *strtok_s( char *string, const char *control, char **context )
{
  char *str;

  if( string == NULL )
    str = *context;
  else
    str = string;

  // Find the next nondelimiter
  while( belong( *str, control ) )
    str++;
  string = str;

  // Find the next delimiter
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) )
    {
       *(str++) = '\0';
       break;
    }
  *context = str;

  if( string == str )
    return NULL;
  else
    return string;
}
```




```
ptr = strtok_s( string, "\",! ", &nextToken );
while( ptr != NULL )
{
   cout << ptr << '\n';
   ptr = strtok_s( NULL, "\",! ", &nextToken );
}
```

```
/*
char *strtok(string, control) - tokenize string with delimiter in
control

Purpose:
        strtok considers the string to consist of a sequence of
        zero or more text tokens separated by spans of one or more
        control chars. the first call, with string specified,
        returns a pointer to the first char of the first token, and
        will write a null char into string immediately following
        the returned token. subsequent calls with zero forthe first
        argument (string) will work thru the string until no tokens
        remain. the control string may be different from call to
        call. when no tokens remain in string a NULL pointer is
        returned. remember the control chars with a bit map, one
        bit per ascii char. the null char is always a control char.

Entry:
       char *string - string to tokenize, or NULL to get next
token
       char *control - string of characters to use as delimiters

Exit:
       returns pointer to first token in string, or
       if string was NULL, to next token
       returns NULL when no more tokens remain.
```

```
char *strtok( char *string, const char *control )
{
  char *str;
  static char *context = string;

  /* Initialize str */

  /* If string is NULL, set str to the saved pointer (i.e., continue
     breaking tokens out of the string from the last strtok call) */
  if( string == NULL )
    str = context;
  else
    str = string;
```

```
  /* Find beginning of token (skip over leading delimiters).
     Note that there is no token iff this loop sets str to point
     to the terminal null (*str == '\0') */
  while( belong( *str, control ) )
    str++;
  string = str;

  /* Find the end of the token. If it is not the end of the string,
     put a null there. */
  for( ; *str != '\0'; str++ )
    if( belong( *str, control ) ) {
       *(str++) = '\0';
       break;
    }

  /* Update nextoken */
  context = str;

  /* Determine if a token has been found. */
  if( string == str )
    return NULL;
  else
    return string;
}
```

國立中正大學103學年度碩士班招生考試試題

3. (6%) The following C function converts a string of lowercase characters into uppercase characters. forexample, it will convert "abc" into "ABC". The parameter **`str`** is the string. Fill in the missing expressions in (l) and (2).

```
void lower2upper(char str []) {
    int i = 0;
    while(   (1)   ) {
        str[i] = str[i] +   (2)   ;
        i++;
    }
}
```

國立中正大學103學年度碩士班招生考試試題

3. (6%) The following C function converts a string of lowercase characters into uppercase characters. forexample, it will convert "abc" into "ABC". The parameter **`str`** is the string. Fill in the missing expressions in (1) and (2).

```
void lower2upper(char str []) {
    int i = 0;
    while( str[i] != '\0' ) {
        str[i] = str[i] +  'A' - 'a' ;
        i++;
    }
}
```